NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE DESIGN OF AN INTELLIGENT
MULTIDISK CONTROL MODULE FOR
VME BUS BASED SYSTEMS

by

Steven L. Brooks

December 1987

Thesis Advisor Larry W. Abbott

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) THE DESIGN OF AN INTELLIGENT MULTIDISK CONTROL MODULE FOR VME BUS BASED SYSTEMS(U)

12 PERSONAL AUTHOR(S)
BROOKS, Steven L.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1987	91

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	VME bus, flexible disk drive, disk controller, concurrent disk operations

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The design of an intelligent multidisk control module for VME bus based systems is presented. The control module is designed to support concurrent disk operations on up to four flexible disk drives with multiple VME bus MASTERS. The design is presented for a UNIX compatible operating system but the operating system interface is kept simple enough that the multidisk control module can be used with most modern operating systems with minimal changes required.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Larry W. Abbott	22b TELEPHONE (Include Area Code) (713) 483-8593 22c OFFICE SYMBOL 62At

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

☆ U.S. Government Printing Office 1986—606-

Approved for public release; distribution is unlimited.

The Design of an Intelligent Multidisk Control Module for VME Bus Based Systems

by

Steven L. Brooks
Lieutenant Commander, United States Navy
B.S., University of Utah, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1987

ABSTRACT

The design of an intelligent multidisk control module for VME bus based systems is presented. The control module is designed to support concurrent disk operations on up to four flexible disk drives with multiple VME bus MASTERS. The design is presented for a UNIX compatible operating system but the operating system interface is kept simple enough that the multidisk control module can be used with most modern operating systems with minimal changes required.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The arrival of the current, more powerful, and faster 16/32 bit microprocessors has enabled the development of high performance desktop systems rivaling main frame systems of ten years ago. These high performance microcomputer systems are capable of supporting multiple users and may consist of several processors working together in systems of mixed capacity and speed. To effectively use the full potential of these high performance microcomputer systems a different architectural approach to system organization must be adopted; one that eliminates the traditional small system "bottle necks".

One of the areas most in need of an improved architecture approach is the control of system input/output (I/O). Maximum performance is achieved when the processors are not kept waiting by slower devices but are allowed to continue processing while the slower devices catch up. This area is particularly important to multiuser systems where total throughput is I/O bound, primarily by how fast user data is exchanged with secondary storage, typically disk drives.

Time lost to waiting is particularly evident in data transfers with flexible disk drives (floppy drives). In the best case, with the disk head over the data to be accessed, a typical floppy drive requires 13 to 27 microseconds to transfer a byte of data, while a typical processor requires less than 1 microsecond to transfer the same byte of data. This amounts to the processor spending at least 90% of the transfer time waiting for the floppy drive. This assumes the traditional direct control of the floppy drive by the processor.

Some new designs attempt to alleviate the floppy I/O wait problem by using direct memory access controllers (DMAC) to do the data transfer. This allows the processor to set up the transfer, then proceed to other tasks instead of waiting for the transfer to be completed. Some examples of this are the new IBM Personal System/2 series systems and some high performance VME bus system modules produced by Motorola, Force Computers and Signetics. The DMAC approach eliminates the need for a processor to wait for a data transfer; however, a "bottle neck" still exists in a multiuser system.

In multiuser systems the user's primary memory space is normally not large enough to meet program and data needs so portions are held in secondary storage, disks, until needed. When a user's process requires access to the disks, that process "sleeps" during the transfer and the processor executes another user's process. The "bottle neck" appears when many processes need access to the disks and the user processes stack up waiting for disk access. This occurs because most desktop systems attach up to four disk drives to a single disk controller, but only one disk drive can transfer data at a time . Thus, even if the required data is on a separate disk, the I O system can only access one disk at a time. A way to improve disk access is to use multiple disk controllers to allow concurrent disk operations. The combination of concurrent disk operations and direct memory access will eliminate the small system I O "bottle neck".

B. DESIGN OBJECTIVES

The objective of this thesis is to develop a hardware kernal of a disk control module (DCM) that incorporates the benefits of direct memory access and concurrent disk operations, as discussed above. The DCM should hide the disk drive command and control requirements by accepting high level commands from the host and translating the host commands into the commands and control signals required by the disk drive. The target environment (host system) is a small, inexpensive but powerful and flexible development system that may begin small and expand to meet user needs. The DCM should be flexible enough to accomodate a variety of common floppy disk drives and be easily integrated into a system. The hardware and software interfaces should be general enough to allow easy migration to any system.

To support these objectives a modular design will be developed to achieve an architecture that can be easily modified to accomodate changes in major interfaces. The major interfaces, host to DCM and DCM to disk drive, will be kept as general and as simple as possible to allow easy migration to a variety of host systems. Migrating to a different bus system, operating system, or changing disk drive type should not require redesigning the DCM but rather just those interfaces directly affected by the changes. This should allow cost and performance tradeoffs to be easily accomodated and changed as system requirements change.

C. HOST SYSTEM

A block diagram of the host system is given in figure 1.1. The host system will consist of one or more processors and additional functional modules for printers, displays, memory, etc. One of the primary attributes of the host system is flexibility; it should be able to accomodate functional modules with varying data path widths (8, 16, or 32 bit), varying access time requirements, and allow the functional modules to be added or removed with minimal adjustments required. The host is assumed to have multiuser capabilities but this is not a requirement; the DCM should benefit single user systems as well as multiuser systems. As shown, multiple DCM's may exist in the host system to accomodate heavy I/O demands.

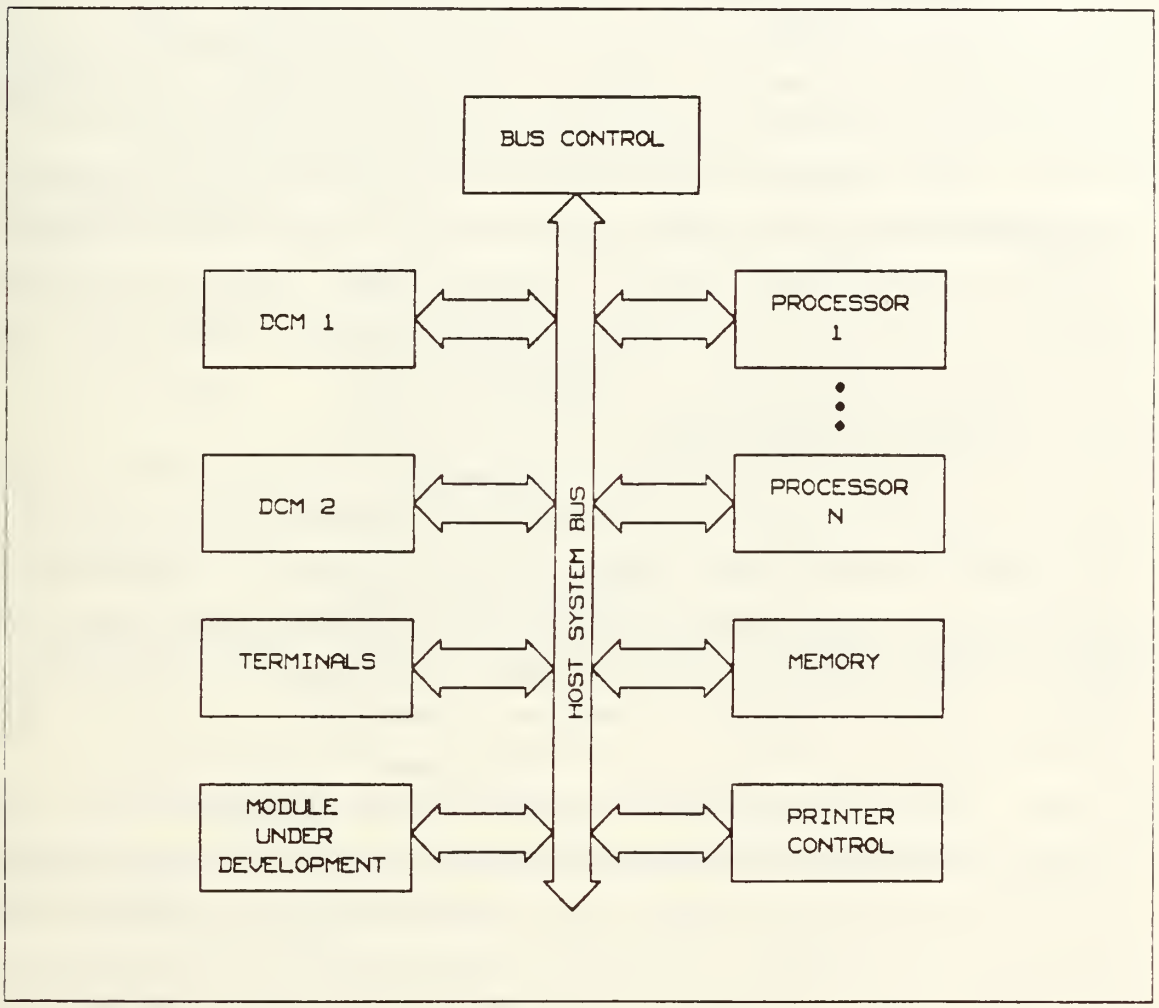


Figure 1.1 Host System Block Diagram.

a. Host Bus Architecture

The host bus architecture is a key element in the flexibility of the system. As indicated earlier, the host system will be composed of modules with varying access times and different data path widths. The ability to support such a mixture of modules is important in allowing older modules to interact with future designs without redesigning the older module's bus interface. The simplest type of bus with the above attributes is an asynchronous bus with support for multiple data path widths [Ref. 1].

A review of currently available buses [Refs. 2,3] reveals two possible bus systems:

- Futurebus (IEEE P896) and
- VME bus (IEEE 1014).

The Futurebus is a very high performance asynchronous bus with multiplexed data paths of 8, 16, and 32 bits. This would have been an excellent choice but the specification is still evolving and is not published for design use yet.

The VME bus is a well established, high performance asynchronous bus with nonmultiplexed 8, 16, and 32 bit data paths. The bus system specification is published, well documented, and has significant silicon support from Motorola, Signetics, and others. The 16 bit data/24 bit address version of the VME bus will be used as the host system bus for the design presented here.

b. Host Operating System

In order to maintain the host's generality and flexibility, the host operating system should be one that is used on a variety of processor types and is easily changed or expanded. This host software interface generality is required to allow the DCM to be designed with relative independence from the host hardware, thus allowing a wider range of host processor types to be accomodated. The flexibility is in keeping with the desire to start small and grow as required. It does little good to have a flexible hardware design if the software can not take advantage of it.

The UNIX operating system fills the above requirements nicely. UNIX hides the machine architecture from the user and runs on a wider range of processor types, from microprocessor systems to main frames, than any other operating system. The DCM can be thought of as a user in this case. Because UNIX is modular and written in a high level language, "C", it is relatively easy to add or change functional software modules. Additionally, UNIX has a simple, consistent interface to peripheral devices and multiuser capabilities. [Ref. 4: pp. 3-4]

D. MAJOR COMPONENT SELECTION

1. Control Processor

The DCM control processor is responsible for executing high level commands from the host and coordinating the data transfers between the host system bus and the attached floppy disk drives. The execution of host commands is essentially a translation process where the terse host commands are expanded into the more detailed command sequences and control signals required by the floppy disk drives. The performance required for the control processor is a function of the tasks assigned by the host system. These tasks may be as simple as fetching a block of data or as complex as acting as the system file manager. In all cases the physical characteristics of the floppy disk drive are hidden from the host and used only by the control processor.

The combination of target host bus system (VME bus), operating system (UNIX), and possible complexity of tasking makes the Motorola MC68000 microprocessor an excellent choice as the DCM control processor. The VME bus was originally designed to support the MC68000 and there are many UNIX systems based on the MC68000 family of microprocessors. Additional supporting attributes are:

- cost - relatively inexpensive, $10 for an 8 MHz version MC68000
- silicon support - extensive family of peripheral support chips available from multiple manufacturers
- compatibility - upward compatible with more powerful members of the MC68000 family; pin-for-pin compatible with the MC68010, affording an easy upgrade path to virtual machine virtual memory operation
- longevity - used in many current microprocessor systems, ensures continued future support
- software support - extensive software support from many vendors; including operating systems, high level language compilers, and utility libraries

2. Direct Memory Access Controller

The MC68000 has three powerful direct memory access controllers as peripheral support chips which are software compatible with each other. They are:

- MC68430 - one DMA channel
- MC68440 - two DMA channels
- MC68450 - four DMA channels

The MC68450 will be used in the DCM design to offer maximum disk drive support. Reduced versions of the DCM would use the MC68430 or MC68440 and a subset of the MC68450 software developed here.

3. **Disk Drives**

The floppy disk drives selected for use are IBM 3740 single density format (FM) and IBM System 34 double density format (MFM) compatible drives. These are the most common and least expensive of the floppy drives available today. They include 8 inch, 5-1/4 inch and 3-1/2 inch form factor drives with formatted capacities of 180 kilobytes to 720 kilobytes per disk. The IBM compatible drives were selected primarily due to cost and availability considerations.

4. **Disk Drive Controller**

Selecting the floppy disk controller was one of the more difficult decisions in the design process. Most manufacturers produce disk control chip sets with various features. The current trend is to combine the controller with the data separator and support circuits, reducing the the disk control function to one or two chips. Even in these reduced count chip sets there is a variety of features available to the system designer.

The Standard Microsystems Corporation (SMC) FDC9268 floppy disk controller was selected for the following reasons:

- format - compatible with the IBM 3740 and IBM System 34 formats, both single and double sided drives
- drives contolled - can control 8 inch, 5-1/4 inch and 3-1/2 inch drives with capacities to 720 kilobytes
- single chip - combines disk control with data separator in a single chip
- cost - relatively inexpensive, $15 each in lots of one
- availability - readily available in large and small quantities
- software - software compatible with the very popular NEC 765/Intel 8272 floppy disk controllers used extensively in personal computer systems

II. PRELIMINARIES

A. OPERATING SYSTEMS

Operating systems act as the interface between the machine hardware and the users. They consist of an organized collection of programs that allocate resources and provide the users with a set of facilities to interact with the hardware.

Operating systems are generally classified into four structures:

- monolithic systems
- layered systems
- virtual machines
- client-server model

Most modern operating systems, regardless of structure, have the control mechanism and higher functions implemented in a higher programming language and accomplish hardware dependent tasks with calls to procedures written specifically to control the hardware. [Ref. 5: pp. 36-43]

Operating systems generally perform all system input output (I O) for the users. The goal of the operating system I O software is to present a simple, consistent, easy-to-use interface to the user. The peculiarities of the hardware are hidden from the user by organizing the I O software as a series of layers, with the lower layers concerned with controlling the hardware, and the higher layers concerned with presenting the easy-to-use interface to the user. [Ref. 5: pp. 116-118]

Layering the I O software also allows the higher layers to have a certain amount of device independence when dealing with specific types of I O devices. An example of this is the operating system interface to secondary storage such as floppy-disks (disks). The basic data structure and data control algorithms are the same for all disks, regardless of manufacturer or type of disk drive. The higher layers of the I O software can implement data management, and the lower layers can adjust for hardware variances. [Ref. 5: pp. 118-120]

The lowest layer of I O software consists of the procedures that interact directly with the hardware. There may be several procedures at this level to deal with the various hardware requirements such as formatting or reading a disk. This hardware specific layer of procedures is common for most modern operating systems and is the simplest common point between operating systems.

The hardware specific procedure layer of the operating system will be the level used to interface the disk control module (DCM) to the operating system. The hardware specific procedures are nothing more than software modules that accept command parameters and data from higher operating system levels and generate the required instruction flow to activate the required hardware signals to accomplish the desired task. Interfacing the DCM at this level means designing the DCM to appear as a software module to the host operating system.

Creating the software module appearance is not difficult. All that is required is memory accessible to the host and DCM for command and data exchanges. This arrangement should work for virtually any modern operating system.

B. UNIX

A complete discussion of the UNIX operating system is beyond the scope of this paper. The UNIX operating system is included here to show that the software interface selected above is satisfactory for interfacing the DCM to a UNIX system. The following discussion is intended to show the hardware dependent software module location in the UNIX architecture, the DCM software interface level, and general interaction with UNIX.

The UNIX operating system can be viewed as a layered operating system. A high-level view of the UNIX architecture is shown in Figure 2.1. The outer most layer represents the user's interface to UNIX. The next layer consists of utility programs such as a text editor and system command modules. The inner most layer of UNIX, surrounding the hardware, is the heart of the UNIX operating system, the kernel.

The UNIX kernel allocates resources to and controls all user processes. The user interacts with the kernal via the utility programs which pass user requirements to the kernel by well defined system calls. A block diagram of the kernel is shown in Figure 2.2.

The UNIX file subsystem uses index nodes (inodes) and inode tables to identify and locate files. Each file has a single inode which contains a description of the disk layout of the file data, logical unit on which the file is located, and administrative data about the file. The file subsystem translates the user's file name into an inode and enters the inode into the kernel inode table indicating that the file is active. Each disk contains an inode list of all the files on the disk, similar to the file allocation table used in MS-DOS.

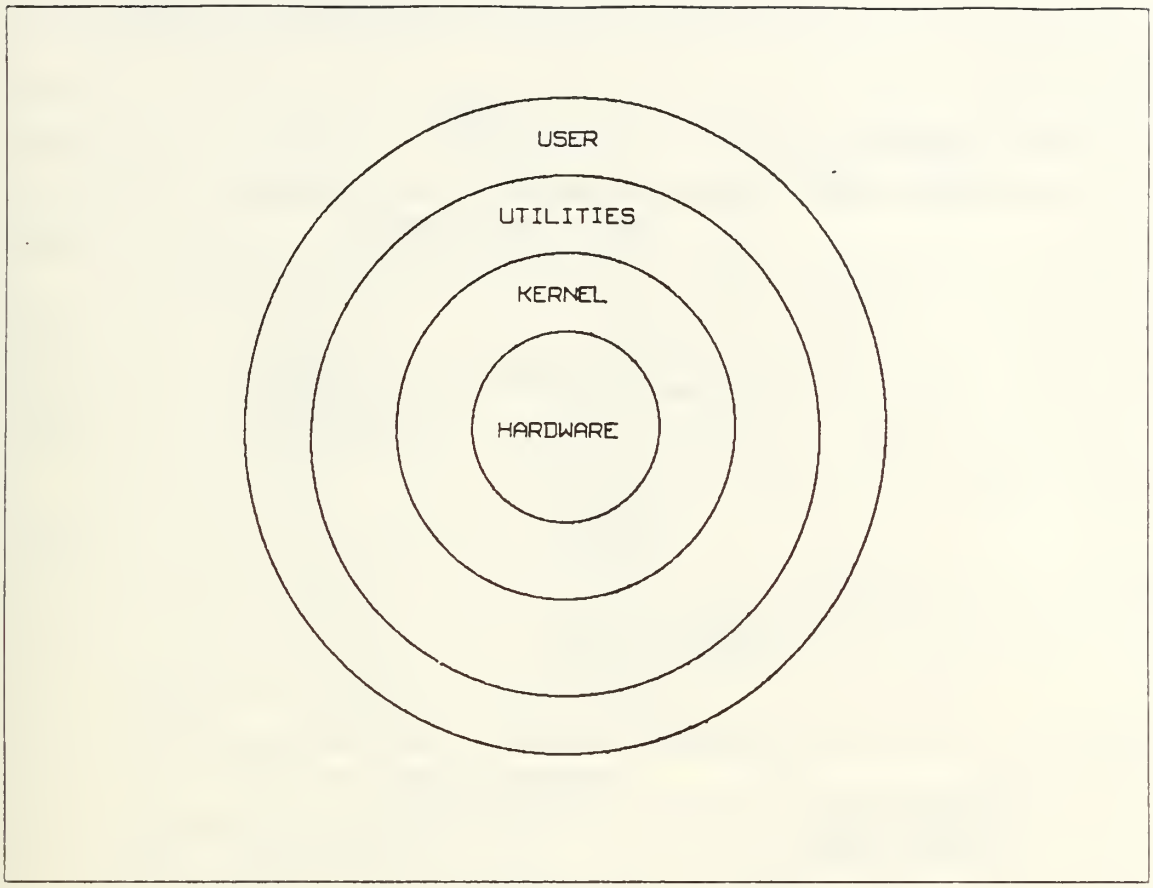


Figure 2.1 UNIX Architecture.

The file subsystem deals with file data on a logical level vice physical disk level. The internal tables for controlling file data manipulations are based on logical device locations. The translation of the logical device location to a physical device location takes place in the device drivers.

The file subsystem has the ability to cache data as it is manipulated. Caching data is used to minimize the frequency of the relatively slow disk accesses by keeping current data resident in kernel memory buffers. New data read from the disk is normally put into buffers for manipulation and then returned to the disk when it is no longer needed. The transfer of data to the buffers is accomplished by the device drivers as directed by the file subsystem.

The device drivers are tailored to particular types of devices. There will be separate device drivers for disks, terminals, magnetic tape, etc. The device driver translates the logical location parameters and operation commands received from the

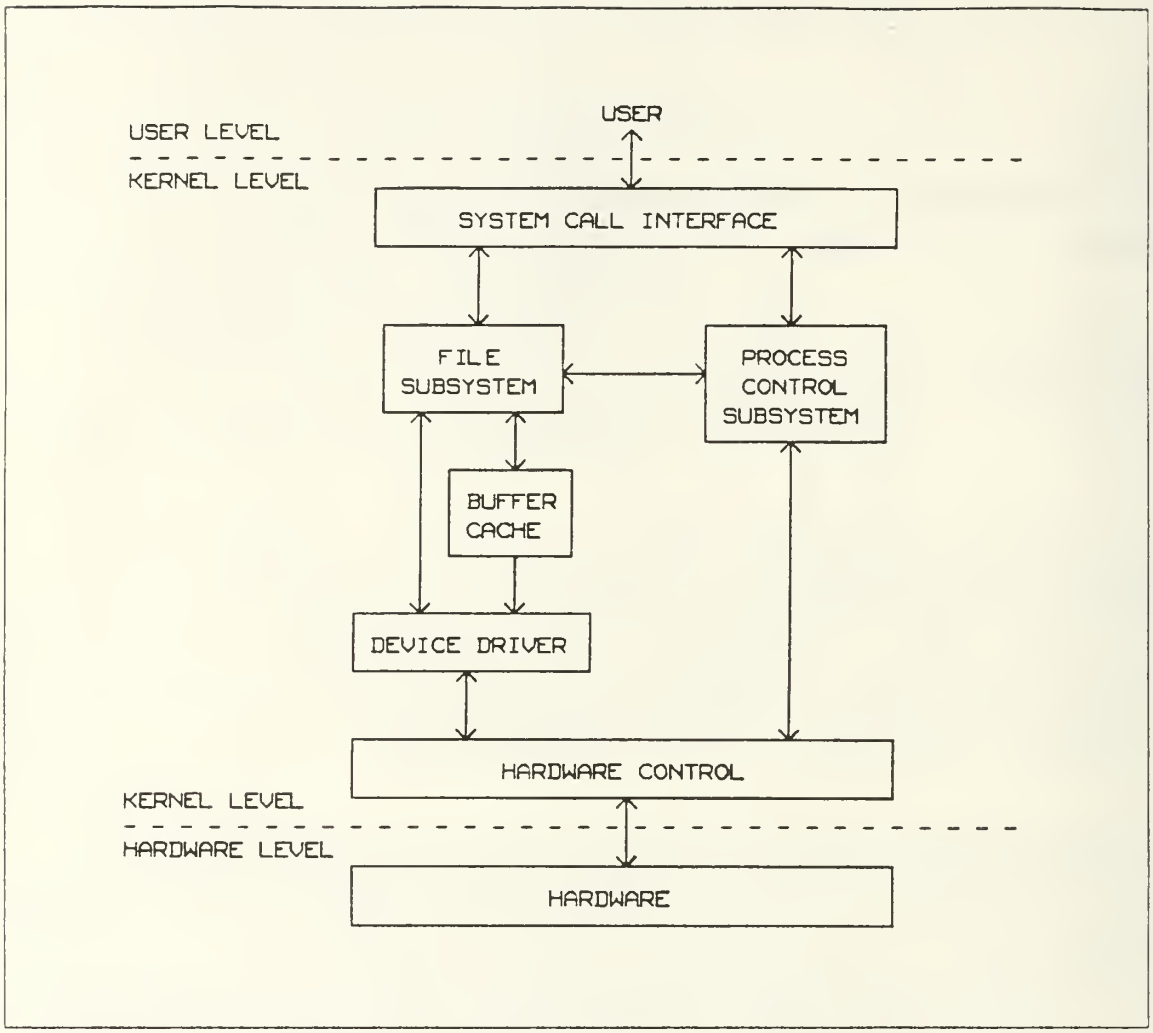


Figure 2.2 UNIX Kernel Block Diagram.

file subsystem into physical locations and operations suitable for the specific type of device to be operated on, disks in this case. The physical location parameters (sector, track, and disk drive number) and the operation to be performed are passed to a disk control subroutine in the hardware control layer. The disk control subroutine is written to carry out the operation on the specific disk drive installed.

The DCM will interface to the UNIX operating system at the hardware control layer by essentially replacing the disk control subroutine. Only minor changes to the device drivers will be required because the DCM will function like the disk control subroutine it replaced. The DCM will appear to the device driver as a software module in memory.

C. VME BUS OPERATION

The VME bus is one of several bus systems available to interconnect data processing, data storage, and peripheral control devices into a closely coupled hardware configuration. The VME bus is an approved IEEE bus standard (IEEE 1014) developed by Motorola, Inc. to support Motorola 68000 microprocessors as a backplane bus.

The VME bus was designed to provide the systems designer with a flexible bus architecture with which to construct microprocessor systems with off-the-shelf hardware and software components. Hardware and software components designed for VME bus applications are available from Motorola, Signetics, Mostek, and others.

The main strengths of the VME bus are best shown by the objectives of the VME bus specification as summarized below:

- to allow communication between devices on the VME bus without disturbing the internal activities of other devices interfaced to the VME bus,
- to specify the electrical and mechanical system characteristics required for reliable and unambiguous communication,
- to specify protocols that precisely define the interaction between the VME bus and the devices interfaced to it, and
- to provide a system where performance is primarily device limited rather than system interface limited.

The elements of a VME bus based system are shown in Figure 2.3. The user devices interface to the VME bus via the functional modules and bus interface logic. The functional modules provide protocol control of the interaction between the VME bus and user's devices, and the interface logic adheres to the specified drive and loading requirements of the interfaced devices. The bus interface logic consists of relatively simple TTL receivers and transmitters because the VME bus drive and timing requirements were designed with these off-the-shelf interface devices in mind.

As seen in Figure 2.3, the VME bus consists of four sub-buses:

- data transfer bus (DTB) - provides data, address, and control signals to allow VME bus MASTERs to direct data transfers between themselves and DTB SLAVES
- arbitration bus - provides a means of transferring control of the DTB between two or more MASTERs in an orderly manner
- priority interrupt bus - provides seven levels of interrupts for interfaced devices to request interruption of normal bus activity
- utility bus - provides signals for timing and coordination of power-up and power-down of VME bus systems

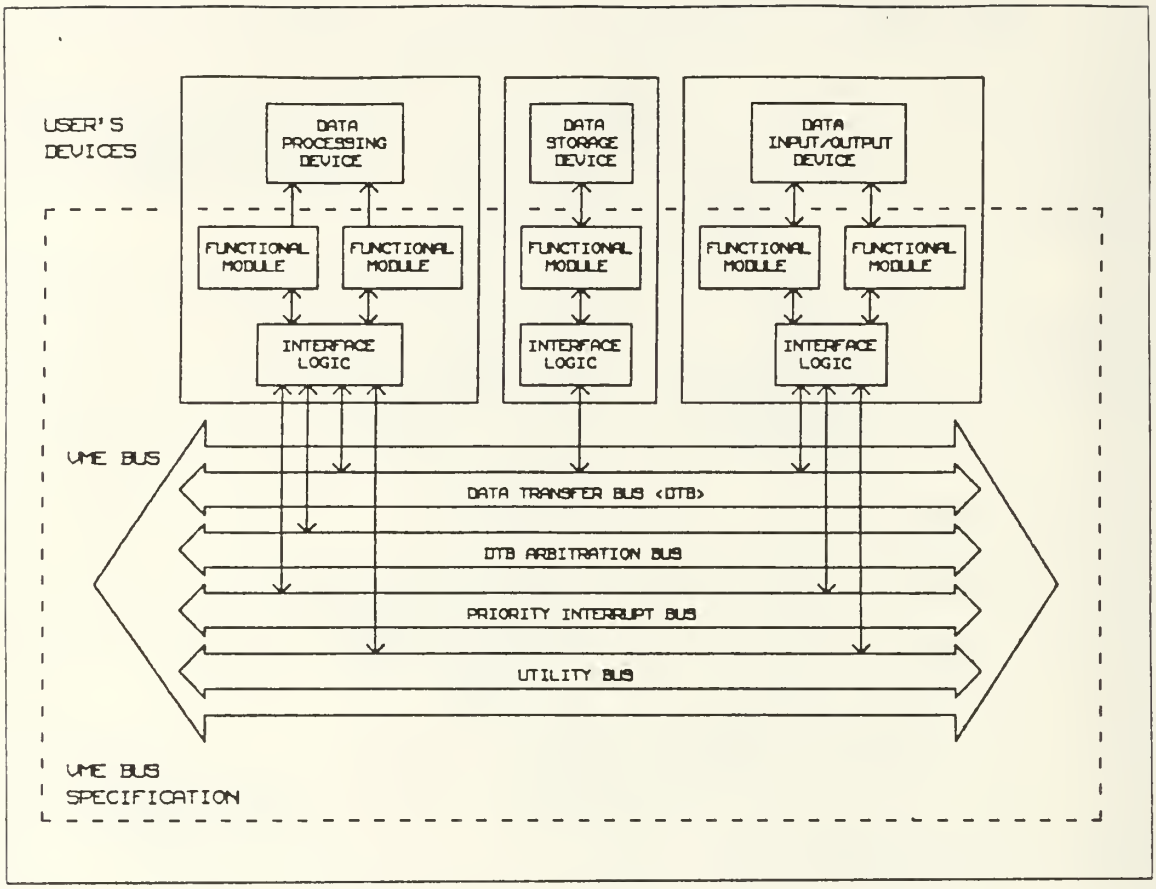


Figure 2.3 VME Bus Elements.

The VME bus protocol is enforced by the functional modules. The VME bus specification defines eleven functional modules to support the various VME bus modes of operation. The primary functional modules of importance to the DCM are:

- MASTER - initiates DTB cycles and controls the DTB in order to transfer data between itself and a SLAVE
- SLAVE - transfers data between itself and a MASTER when selected to participate in DTB cycles
- INTERRUPTER - generates an interrupt request on the priority interrupt bus and then provides status/vector information to the INTERRUPT HANDLER when requested
- INTERRUPT HANDLER - detects interrupt requests on the priority interrupt bus and responds to these requests by asking for status/vector information from the INTERRUPTER

The SLAVE interface to the VME bus is the simplest of all functional modules that transfer data over the VME bus. The SLAVE functional module does not initiate or control data transfers; it can be thought of as a memory module with additional decoding logic.

The DCM will be designed as a VME bus SLAVE with interrupt capability. This means two functional modules will be used; SLAVE and INTERRUPTER. This is the minimum configuration that provides a memory-like appearance and has interrupt capability.

The SLAVE functional module can be designed as an 8-bit, 16-bit, or 32-bit data device with 16-bit, 24-bit, or 32-bit addresses. The DCM will be designed as a 16-bit data device with 24-bit addresses. The INTERRUPTER functional module can be designed to request interrupts on any one or all seven interrupt lines. The DCM will be designed to interrupt on any of the seven interrupt request lines. The interrupt level will be software selectable by the host system. In the above configuration, the DCM does not need to interface to the arbitration bus.

As a SLAVE, the DCM must monitor or generate the following VME bus signals:

- LWORD* - designates a 32-bit data transfer request, monitored by SLAVEs
- D00-15 - bidirectional data lines
- DS0* - lower data strobe (same as 68000 LDS*), monitored by SLAVEs
- DS1* - upper data strobe (same as 68000 UDS*), monitored by SLAVEs
- R/W* - read/write signal (same as 68000 R/W*), monitored by SLAVEs
- AM0-5 - address modifiers ,monitored by SLAVEs
- A01-23 - 23-bit address lines ,monitored by SLAVEs
- AS* - address stable signal (same as 68000 AS*), monitored by SLAVEs
- DTACK* - data transfer acknowledge (same as 68000 DTACK*), generated by SLAVEs
- BERR* - bus error signal (same as 68000 BERR*), generated by SLAVEs

The above VME bus signals function the same as their 68000 memory reference counterparts with two exceptions, LWORD* and AM0-5.

LWORD* is used in data transfers with 32-bit devices only. As a precaution, all devices must monitor LWORD* and must respond with a bus error (BERR*) or not respond at all, which causes the VME bus timer to assert BERR*, if the selected device can not provide 32-bit data transfers.

AM0-5 are address modifier lines that specify the type of addressing used in a data transfer: short (16-bit), standard (24-bit), or extended (32-bit). AM0-5 also indicates the type of transfer requested by differentiating between supervisory and non-priviledged data, program, and block transfers.

The DCM will not respond to LWORD* transfers, causing the VME bus timer to assert BERR*. The DCM will respond to supervisory and non-privileged data and block transfers but will generate a bus error for any program transfers such as reading the DCM memory as program memory in instruction fetches.

The basic data transfer capabilities specified by the VME bus data transfer protocol and supported by the DCM are:

- byte transfers - even or odd single byte transfers
- word transfers - single 16-bit transfers
- read-modify-write - 8-bit or 16-bit indivisible read followed by a write to the same address
- block transfer - up to 256 sequential bytes transferred with only the starting address specified

The byte, word, and read-modify-write transfers operate the same as in the standard 68000 memory reference protocol.

In block transfers the MASTER provides the starting address at the beginning of the transfer. The SLAVE latches the starting address in a counter and increments the address as the data strobes change. The starting address is provided only once and incremented by the SLAVE each time the data strobes are negated. The address stable (AS*) is asserted during the entire block transfer and AM0-5 are encoded to specify a block transfer is in progress. The block transfer mode is the fastest data transfer mode on the VME bus because the address propagation and decoding delays are encountered only at the beginning of the block transfer.

The INTERRUPTER functional module follows the standard 68000 interrupt request-acknowledge protocol. The INTERRUPTER generates an interrupt request on one of the seven interrupt request lines (INTRQ*1-7) and waits for an acknowledge. The INTERRUPT HANDLER for the asserted interrupt request line responds by requesting control of the data transfer bus, and after gaining control, asserts interrupt acknowledge (IACK*) to all devices and sends an interrupt acknowledge (IACKIN*) down the interrupt acknowledge daisy-chain. The INTERRUPTERs without active interrupt requests at the level being acknowledged pass the IACKIN* down the daisy-chain via IACKOUT* which becomes the IACKIN* of the next INTERRUPTER in

the daisy-chain. The first INTERRUPTER in the daisy-chain with an active interrupt at the level being acknowledged stops the IACKIN* from propagating further down the daisy-chain and returns an 8-bit vector to the INTERRUPT HANDLER via the data transfer bus. The interrupt level being acknowledged is encoded in address lines A1-3. The IACK* signal is used to indicate to all other devices on the VME bus that an interrupt acknowledge cycle is in progress which means only address bits A1-3 are valid.

D. DISK DRIVES

Floppy disk drives (disk drives) are block oriented mass storage devices. The data is stored as blocks in sectors on the disk. The recording surface is organized as a series of circular tracks broken up into equally sized sectors. Sector data holding capacity varies from 128 bytes to 4096 bytes. The sector is the smallest addressable data block on a disk drive.

There are three standard disk sizes: 8 inch, 5 1/4 inch, and 3 1/2 inch diameters. The 8 inch disk is the oldest version and rarely used today. The 5 1/4 inch disk is currently the most common and least expensive, but the newer 3 1/2 inch disks are gaining in popularity due to their ruggedness, compact size, and higher density.

The data and control fields on a disk are organized into a specific format. There are numerous formats available, but the most common are IBM 3740 (single density) and IBM System 34 (double density) compatible. The data and control fields of a disk track in IBM 3740 format are shown in Figure 2.4.

The sector format consists of the sector address (track sector ID) which identifies the track, sector, side, and length of the sector. The sector is accessed by stepping to the proper track and reading addresses until the desired address is read. The sector address is followed by a cyclic redundancy check (CRC) as an error check on the address. The ID gap (GAP2) provides time for the disk controller to compute the CRC for the address read and compare with the CRC read from the disk to ensure a valid disk sector address has been read. The sector length field contains the number of bytes written in the sector, not the capacity of the sector; a 128 byte sector with only 100 bytes written in the the sector will have a sector length of 100. The sector address field is followed by the data field with a data CRC error check. The post data gap (GAP3) provides time for the disk controller to check the data's CRC plus an additional buffer space to ensure sectors do not overlap due to variances in timing or disk rotation speed

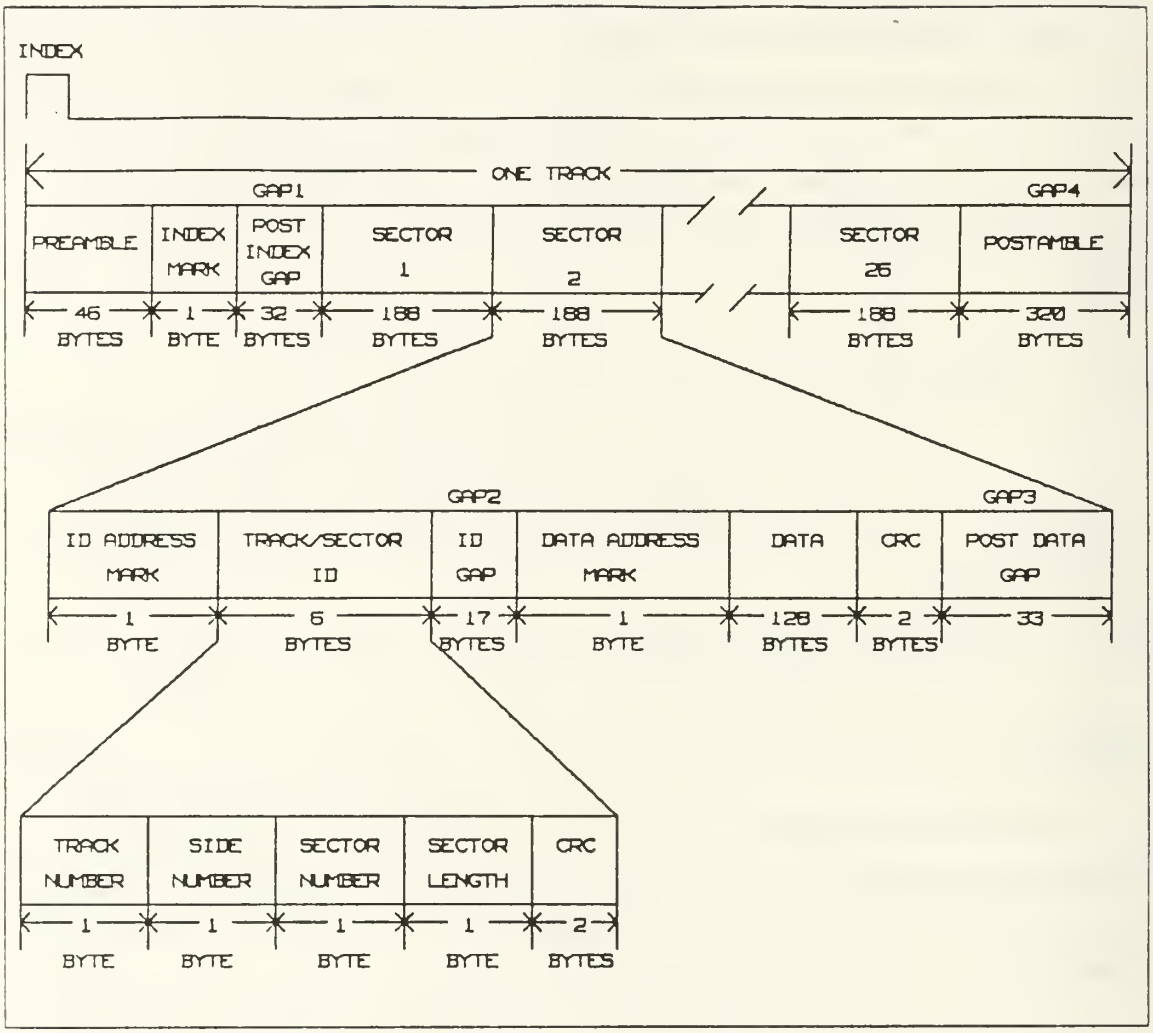


Figure 2.4 IBM 3740 Disk Format.

The IBM format uses special control characters to mark the beginning of some fields. The index mark, ID address mark, and data address mark are special characters that can not be written in normal data format. In these special characters some of the clock pulses are omitted to create unique codes that can not be duplicated in data. The special characters are written only during the disk format operation.

The basic operation of the disk drive is relatively simple. The disk controller instructs the disk drive to move (seek) the read/write head (head) to a specific track and loads the head, i.e. puts the head in contact with the disk. The disk drive starts passing the information from the disk to the disk controller where the information is checked for special characters, sector addresses, and data. When the operation is

complete, the disk controller unloads the head and waits for the next operation. Not all disk drives follow the above sequence exactly because newer disk drives have more capabilities, but the sequence of steps demonstrates the basic disk drive operation.

Disk drives require a variety of signals to be exchanged with the disk controller to accomplish head movement (seeks) and read write operations. The types of signals exchanged are a function of drive type and age. Older 8 inch disk drives require more signals than newer 5 1/4 inch drives. To retain the flexibility of using a variety of disk drives the disk controller must be capable of exchanging a full range of signals that can be customized for the disk drive that is actually installed. The following signals are representative of the signals exchanged between disk controllers and disk drives, regardless of size or type. All signals are asserted low unless otherwise specified.

- unit selects (US1*,...,US4*) - select one of four disk drives
- head load (HDL*) - instructs the disk drive to put the head on the disk surface
- low current track (LCT*) - notifies the disk drive that the head is above track 43 so that precompensation can be used if needed
- fault reset (FR*) - resets the disk drive's fault indicator
- write protect (WP*) - notifies the disk controller that the disk is protected and not writable
- fault (FLT*) - notifies the disk controller that the disk drive has a fault
- ready (RDY*) - notifies the disk controller that the disk drive is ready
- index (IDX*) - index timing mark from the disk drive
- raw read data (RRD*) - composite read data from the disk drive
- write enable (WE*) - instructs the disk drive to write
- write data (WDOUT*) - data to be written on the disk
- head select (HD*) - for two-sided disks, low selects head 0 and high selects head 1
- double density mode (MFM*) - instructs disk drives capable of single and double density modes to use double density mode
- direction (DIR*) - tells the disk drive the direction to move the head in response to a step during seek operations, low = in and high = out
- step (STP*) - instructs the disk drive to move the head one track in the specified direction
- two sided (TS*) - notifies the disk controller that a two-sided disk is installed
- track 00 (TR00*)- notifies the disk controller that the head is at track 00

As indicated in Chapter I, the IBM compatible 5 1/4 inch disk drives will be used in the DCM design. These drives were selected primarily because they are readily available and inexpensive.

III. HARDWARE DESIGN

Previous chapters defined the interfaces and interaction of the disk control module (DCM) with the host system and disk drives. This chapter will present the internal architecture and hardware design created to meet the requirements of the hardware and software interfaces.

A. ARCHITECTURE

A modular architecture will be used to accomodate changes in major external interfaces with minimum perturbation of the DCM realization. The foundation of the architecture is a microprocessor-based central control unit (CCU) which provides overall control and coordination of the DCM operation. The CCU receives commands from the host via the host interface and translates these commands into the signals required, at the disk interface, to accomplish the required disk action. The external interfaces and major component systems of the DCM are shown in Figure 3.1.

1. Host to DCM Interface

The interface to the host system consists of three parts:

- bus control - hardware interface to control DCM responses to host bus activity
- host control - software interface for command and status exchange with the host
- data buffers - software interface for data exchange with the host

The bus control section of the host interface enforces the host bus system physical protocols by providing physical control to the software interfaces.

In Chapter II, the host software interfaces to the DCM were shown to be reflected as memory locations in the host's global memory. The host operating system views the DCM as a software module with specified command parameter and data buffer memory locations. To the host, the DCM appears and functions much like a COMMON or global memory area in a program.

The software module appearance is important to achieving the desired interface generality and flexibility. The host views the DCM as a simple block of memory, this view is consistent for all programs using the DCM and allows a broad range of operating systems to be accommodated. Flexibility is achieved by modifying the bus control section to comply with the selected host bus system protocol for simple memory reads and writes. This preserves the appearance of the software interfaces as the bus system protocols change.

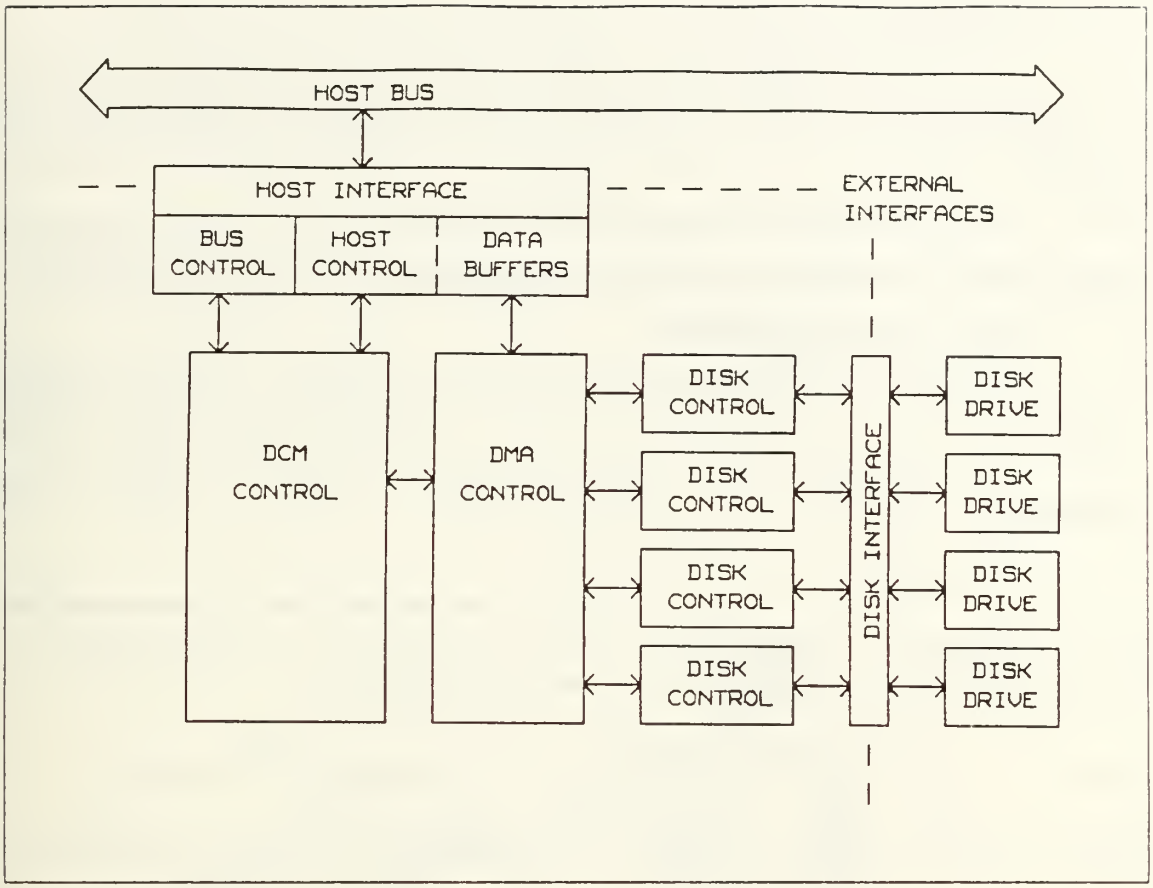


Figure 3.1 Generalized DCM Block Diagram.

Incorporating the software interfaces in the host bus interface section and treating them as buffer memories isolates the DCM internal control functions from the host. These software interfaces also appear to the DCM as a block of memory, analogous to the view held by the host. This memory appearance provides consistent views of the host and DCM with respect to each other.

This organization of the host to DCM interface is used to isolate the internal DCM control functions from the host system and to provide consistent views of the software interfaces as seen by the host operating system and internal DCM control software. The isolation of the DCM control functions provides relative autonomy to the DCM while it is performing required tasks and allows the DCM internal architecture and components to be changed without the knowledge of the host, provided the host operating system view of the software interfaces is not altered.

2. DCM to Disk Interface

As seen in Chapter II, the disk interface is relatively simple. The interface required for one disk drive is simply replicated for each drive to be used, four in this case.

The disk interface consists of TTL type receivers and transmitters to drive control and data lines under the control of the disk controllers onboard the DCM.

3. DCM Internal Architecture

Figure 3.2 is a more detailed block diagram of the DCM reflecting the specific target host bus, the VME bus, and the host interface memory buffers. Internal interfaces between major onboard sections are shown to illustrate a conflict in internal bus usage.

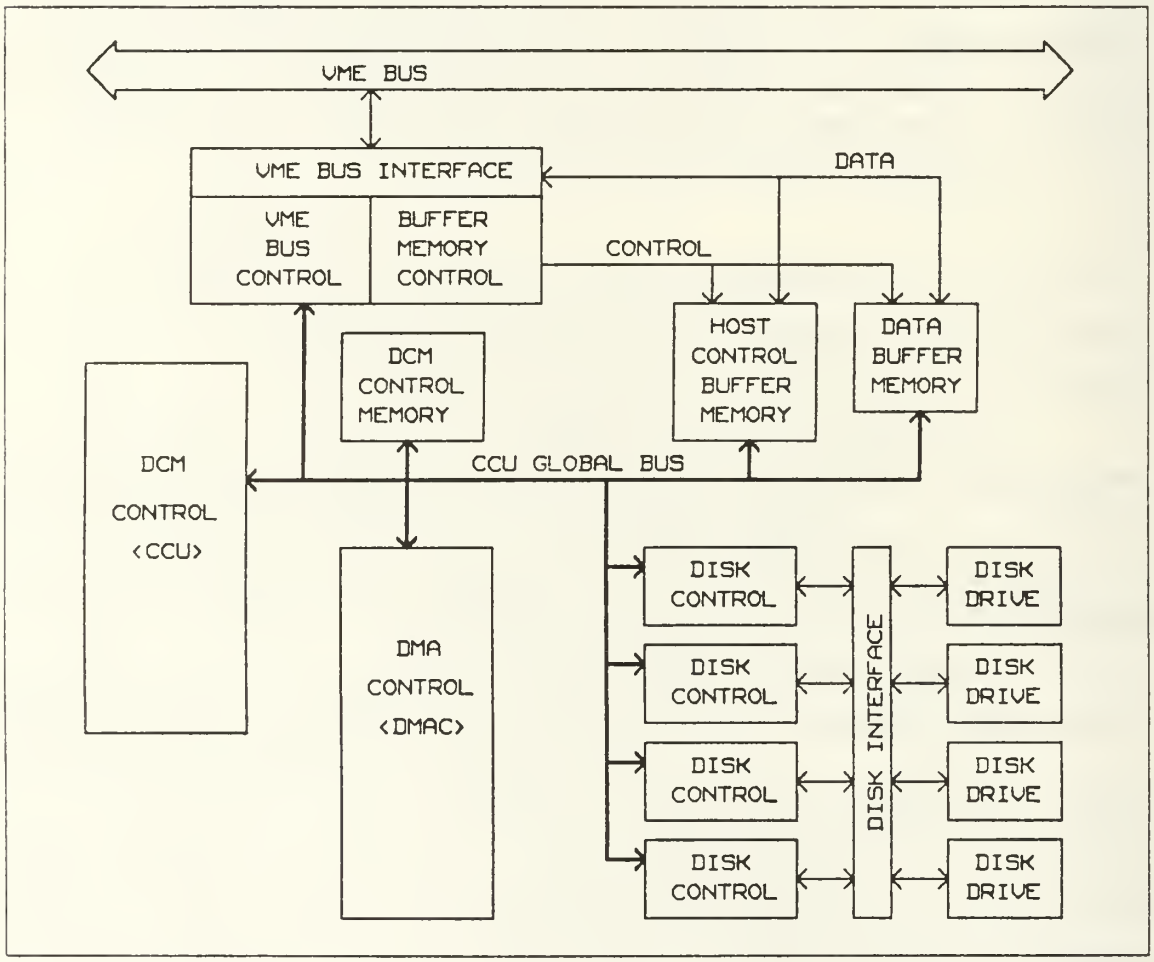


Figure 3.2 DCM Internal Bus Contention.

As indicated earlier, the DCM will be controlled by a microprocessor system. Typical microprocessor systems have a single global bus connecting all components to the microprocessor; the system in shown in Figure 3.2 has the same property. This single bus system may lead to a conflict in the DCM application because two component systems, CCU and DMAC, will be competing for use of the bus. This bus contention arises any time a DMA operation is in progress and the CCU tries to access the bus for program memory references or to set-up a disk controller for a data transfer. This bus contention will degrade system performance, especially with high DMA usage. The contention can be reduced by splitting the CCU global bus into two buses as shown in Figure 3.3.

This dual bus arrangement eliminates bus contention during CCU communication with the host control buffer memory by separating the data flow path and host command path into two sections:

- CCU local bus - accessed only by the CCU, connects CCU memory and those components needed for host command receipt and not needed for actual data transfer
- global bus - shared by the CCU and DMAC, connects those components involved in data transfers

This bus arrangement allows host commands to be received and interpreted without bus contention.

Bus contention will still occur when the CCU attempts to communicate with the DMAC or disk controllers during DMA operations, but the impact can be minimized by allowing the CCU and DMAC to share the global bus on a cycle-by-cycle basis. This bus sharing can be prioritized to ensure that disk data overrun does not occur.

The final architecture, incorporating the features described in preceding discussion, is given in Figure 3.4. The five major sections are summarized as:

1. Host Interface
   a. controls all communication between the host and DCM
   b. appears to the host and DCM as shared memory
2. DCM Control
   a. microprocessor system controlling overall operation
   b. shares global bus with DMAC
3. Global Bus Control
   a. arbitrates global bus access
   b. prioritized to ensure disk data overrun does not occur

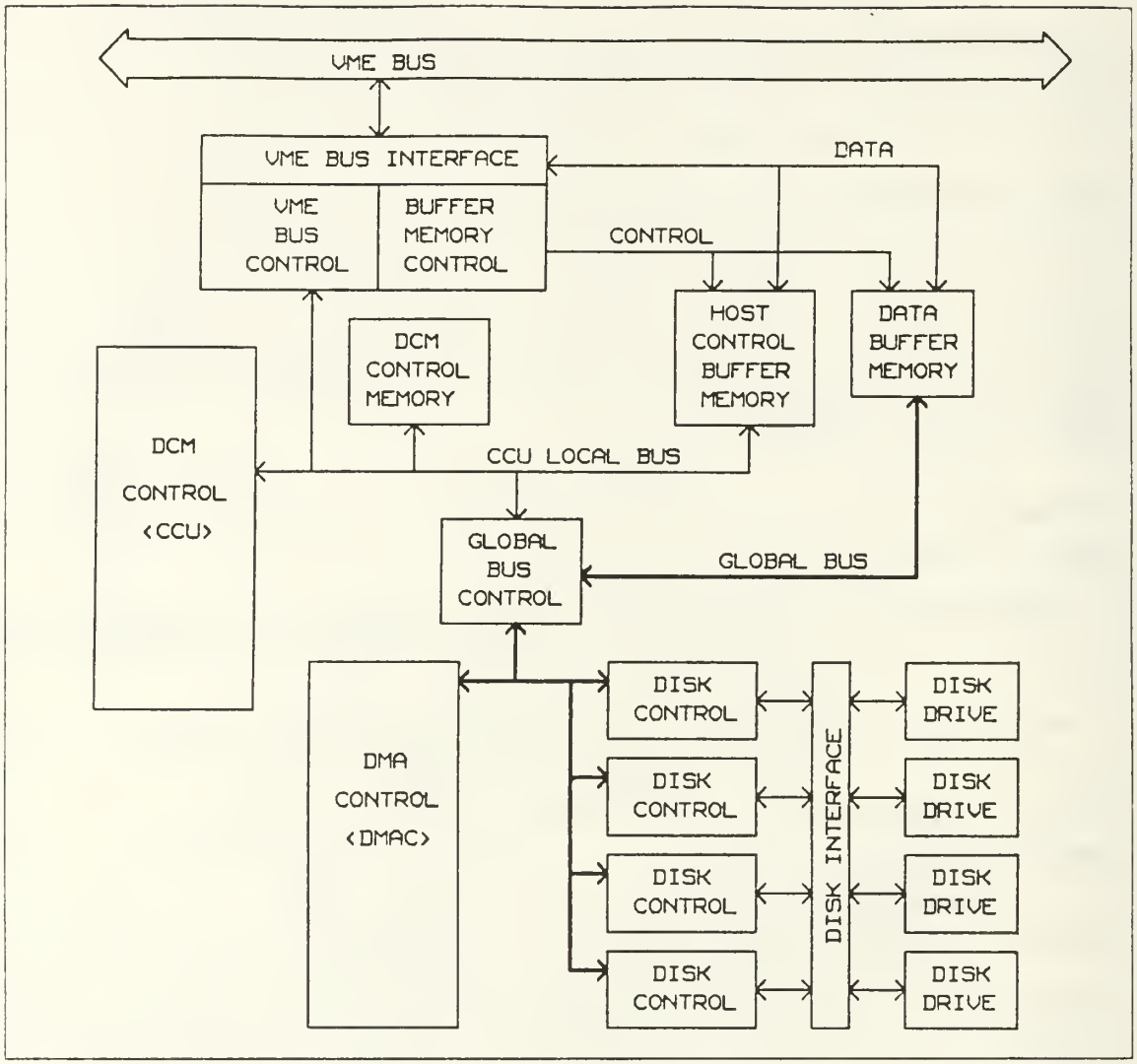


Figure 3.3 DCM Dual Bus Configuration.

4. DMA Control
   a. controls data transfer between disk controllers and data buffer memory
   b. primary user of global bus
5. Disk Control
   a. disk drive interface
   b. controls basic disk drive operation

The remainder of this chapter presents the hardware realization of the above sections. The diagrams used may consist of functional blocks representing groups of devices. Details of the functional blocks are provided in Appendix A.

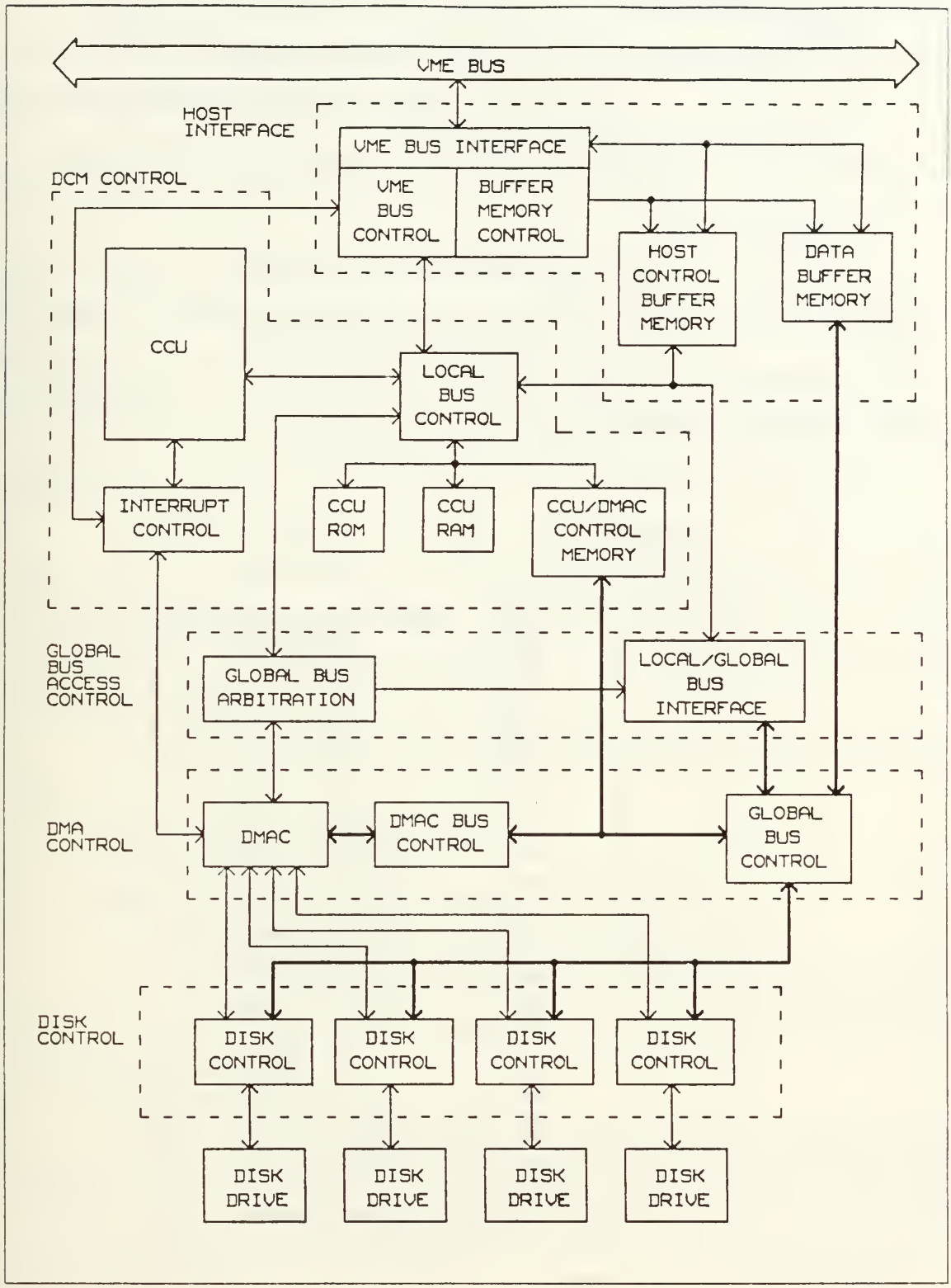


Figure 3.4 Detailed DCM Block Diagram.

B. HOST INTERFACE

The host interface will be treated as two general sections:

- software interface - the memory model of the DCM as seen by the host operating system, and
- hardware interface - hardware required to present the software interface to the host.

1. Software Interface

Figure 3.5 provides the memory map of the DCM as viewed by the host operating system. The memory space presented to the host is an 8 kilobyte block organized as 16 bit words with addressing to the byte level in typical 68000 style memory organization. Host address bits A13 thru A23 are user selectable to allow this block to be mapped anywhere in the host's 16 megabyte address space.

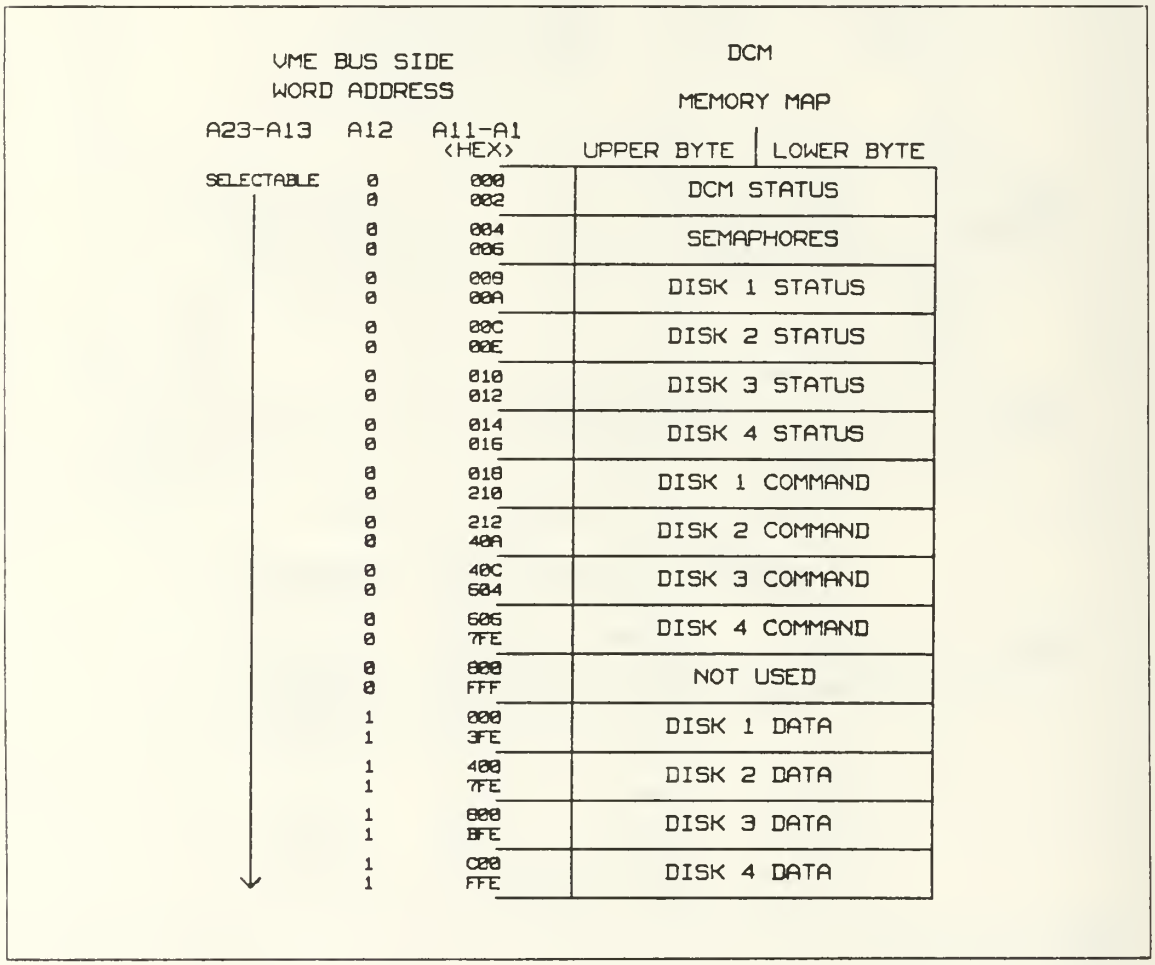


Figure 3.5 Host View of DCM Memory Map.

The DCM memory block visible to the host consists of the host control and data buffer memories of Figure 3.4. These memories are partitioned into smaller functional blocks for the following uses:

- DCM status - four bytes representing the overall status of the DCM
- semaphores - one byte per disk indicating the availability of a disk for use (four bytes total)
- disk status - four bytes per disk indicating the status of the last operation performed by a disk (16 bytes total)
- disk command - 505 bytes per disk for host commands plus one byte for command present indicator (2024 bytes total)
- not used - two kilobytes reserved for command space expansion
- disk data - one kilobyte per disk for disk data storage (four kilobytes total)

The details of the partitioned block formats will be presented in Chapter IV with the software development.

The command present byte in each disk command space is used to notify the DCM that the host has an active command present for the indicated disk. This is accomplished by generating an interrupt to the CCU when the host writes to the last byte location in the specified disk command space. The interrupt generation is totally transparent to the host.

The memory operations that are supported in the DCM host shared memory are:

- single byte (upper or lower) read/write
- word (double byte) read write
- block word read write on blocks up to 256 bytes long
- uninterruptible read-modify-write cycle for semaphore support

Only the details required for hardware implementation of the shared memory are presented above. The format and use of the software interface will be presented in more detail in Chapter IV.

2. **Hardware Interface**

Figure 3.6 is an expanded view of the major sections of hardware used to implement the software interface. The host control buffer memory and data buffer memory are combined into a single dual ported buffer memory module for simplicity. Each major block will be described below.

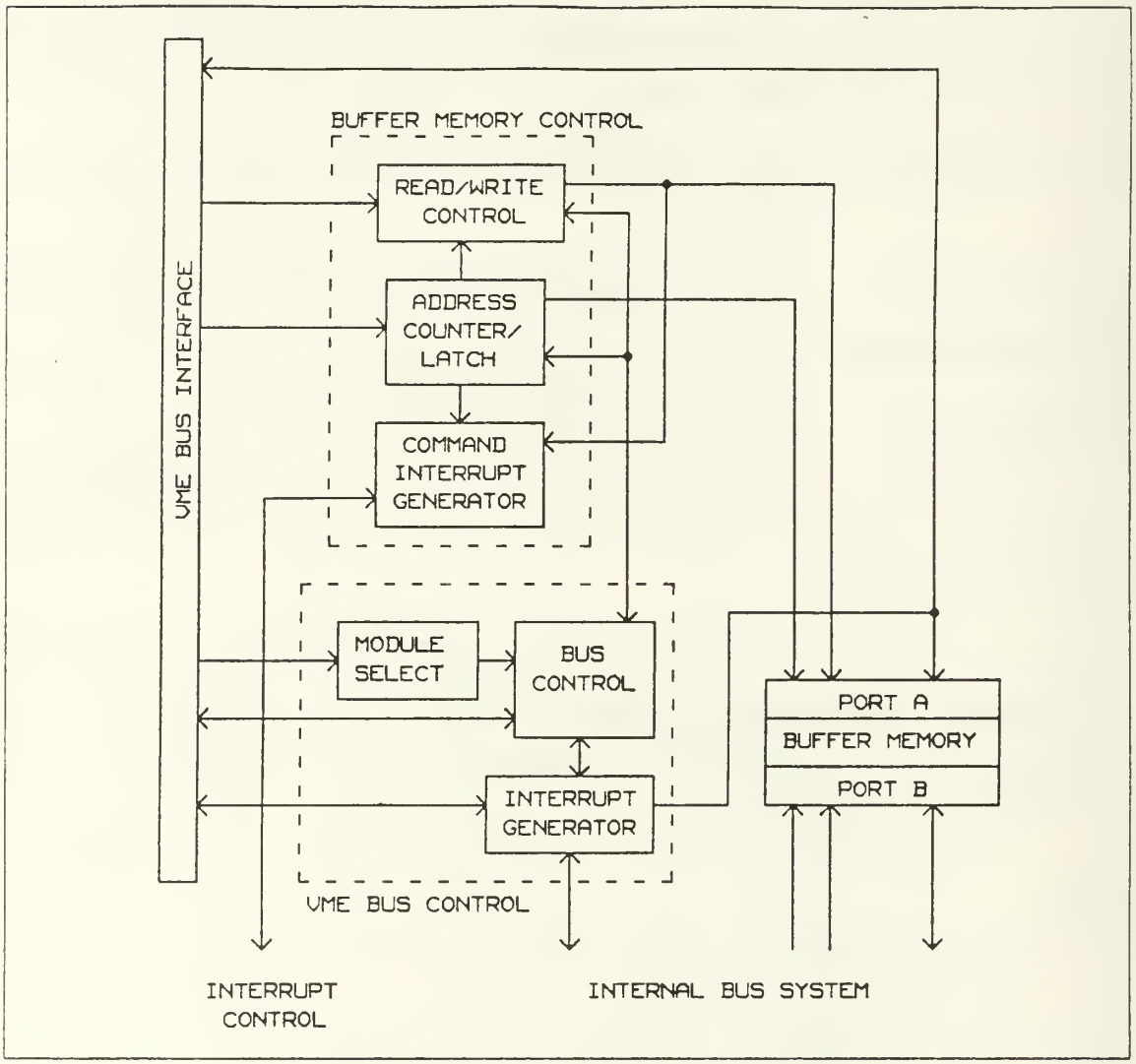


Figure 3.6 Host Interface Block Diagram.

a. VME Bus Interface

Figure 3.7 shows the signals exchanged between the DCM and VME bus. The devices shown for driving or receiving the signals are those recommended in the VME bus specification. The following signals are connected to special purpose devices designed specifically to generate or receive them in VME bus applications:

- BERR*
- DTACK*
- IACK*
- IACKIN*
- IACKOUT*

The special purpose devices are used in the bus control and interrupt generator sections of Figure 3.6 and will be discussed in the design of those sections.

b. Buffer Memory

The host control and data buffer memories are implemented as dual ported memories with host access via port A and DCM access via port B. Figure 3.8 shows port A of the buffer memories which implements the memory map of Figure 3.5. The host control and data buffer memories are separate, byte addressed, dual ported memory banks. The port B operation of these memories will be discussed as part of the CCU and DMAC memory designs.

There is a wide range of devices available for implementing the dual ported memories, from dual ported dynamic RAM controllers to very fast static RAM devices with onchip arbitration logic. The devices used in this design are Integrated Device Technology, Inc. (IDT) static RAMs with onchip arbitration and a wait signal.

The IDT devices have an excellent speed range, from 25 nanosecond to 120 nanosecond cycle time, and onchip arbitration that allows both ports simultaneous access to the memory matrix as long as the addressed locations are not the same. If both ports attempt access to the same location, the winning port gains access and the losing port has the wait signal asserted indicating a delay in access. The speed range is desirable to allow the DCM to provide a range of memory performances to support desired VME bus throughput versus cost tradeoffs with minimal changes in the hardware.

The host control buffer memory is composed of two devices:

- IDT7130 - 100 nanosecond cycle time, organized as 1024 by 8 bit memory locations with onchip arbitration and wait signal
- IDT7140 - 100 nanosecond cycle time, organized as 1024 by 8 bit memory locations and shares the IDT7130 arbitration logic

These devices are designed to work as master (IDT7130) and slave (IDT7140) to minimize cost and complexity. The IDT7130 controls the arbitration and drives the wait signal for both devices.

The data buffer memory is also composed of two devices:

- IDT7132 - 2048 by 8 bit version of the IDT7130
- IDT7142 - 2048 by 8 bit version of the IDT7140

The operation of these devices is the same as for the host control buffer memory.

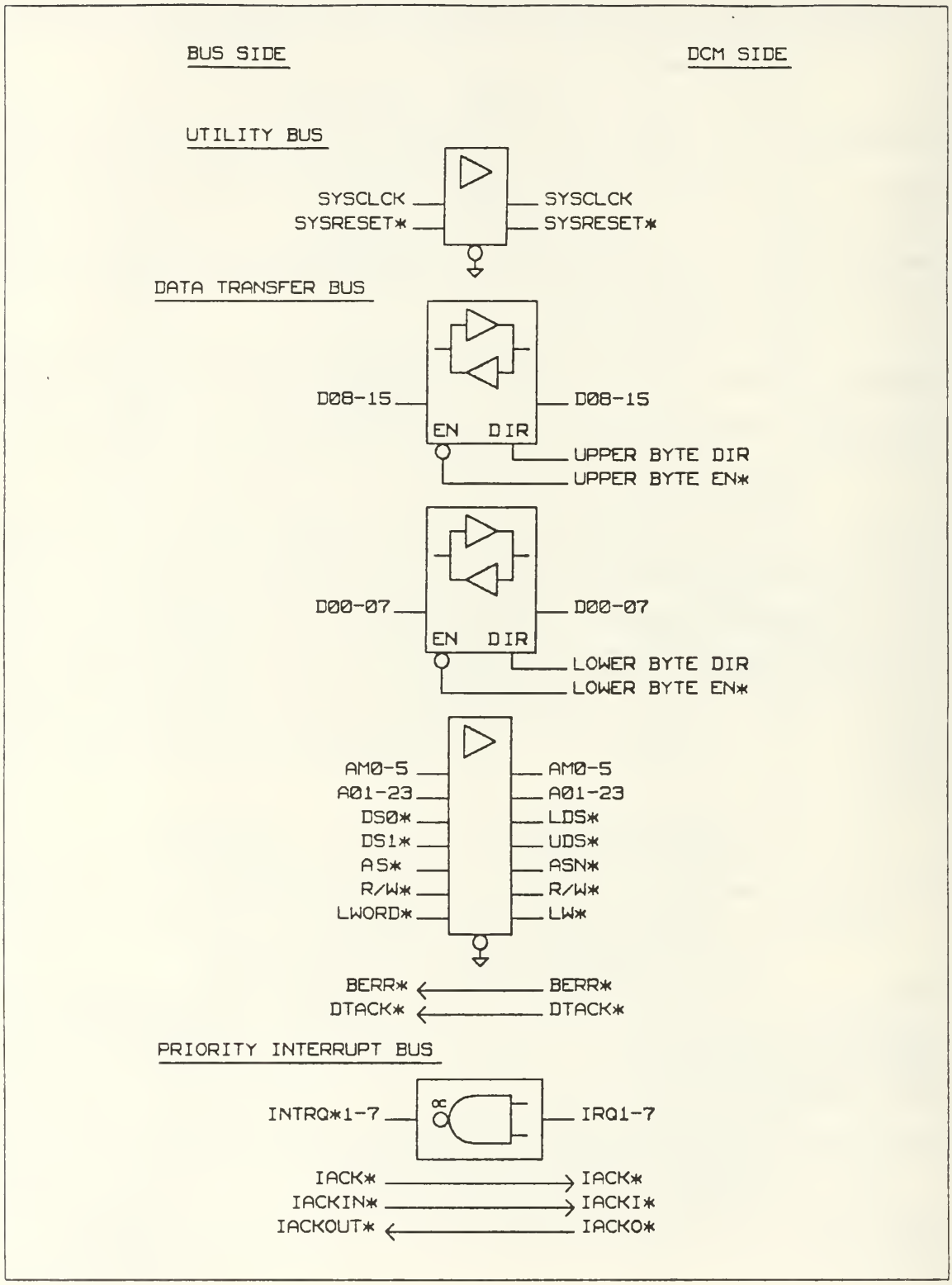


Figure 3.7 VME Bus Interface.

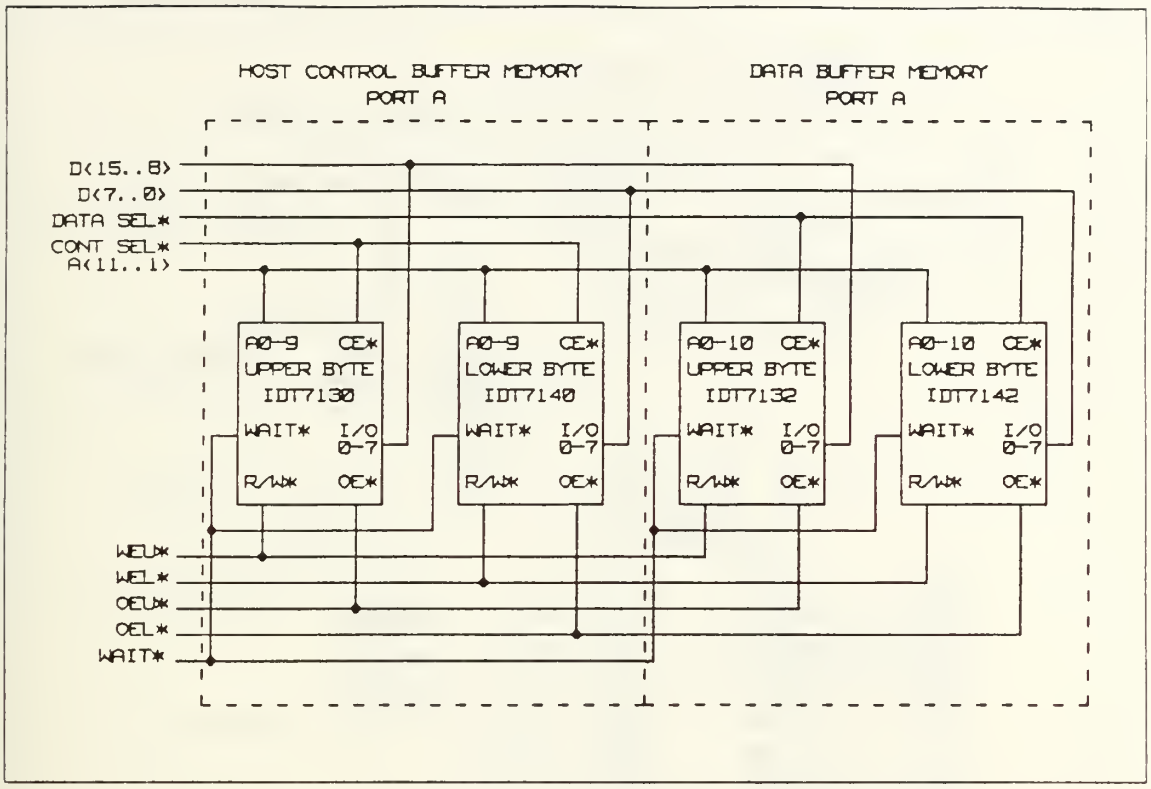


Figure 3.8 Buffer Memory Port A.

c. Module Select

The module select circuit, Figure 3.9, is used to decode address bits A13 thru A23 and the address modifier codes AM0 thru AM5 to determine if the current VME bus memory cycle is intended for this DCM.

The DCM will respond to standard address single byte word and block word data transfers as described in Chapter II. Program accesses, such as attempting to read the DCM buffer memories as instructions, will result in no response and cause the bus timer to time out and assert the bus error signal.

The module select circuit generates two signals required by other circuits. The module select signal (MODULE SEL*) indicates the user selected address switches match A13 thru A23 of the VME bus and the address modifier lines are set to standard address data transfer. The MODULE SEL* signal notifies the bus control section that the current memory cycle is for this DCM.

The block transfer signal (BLT) results when the address modifier lines are set for a block transfer. This signal enables the address counter circuit to increment the memory address as the data strobes change.

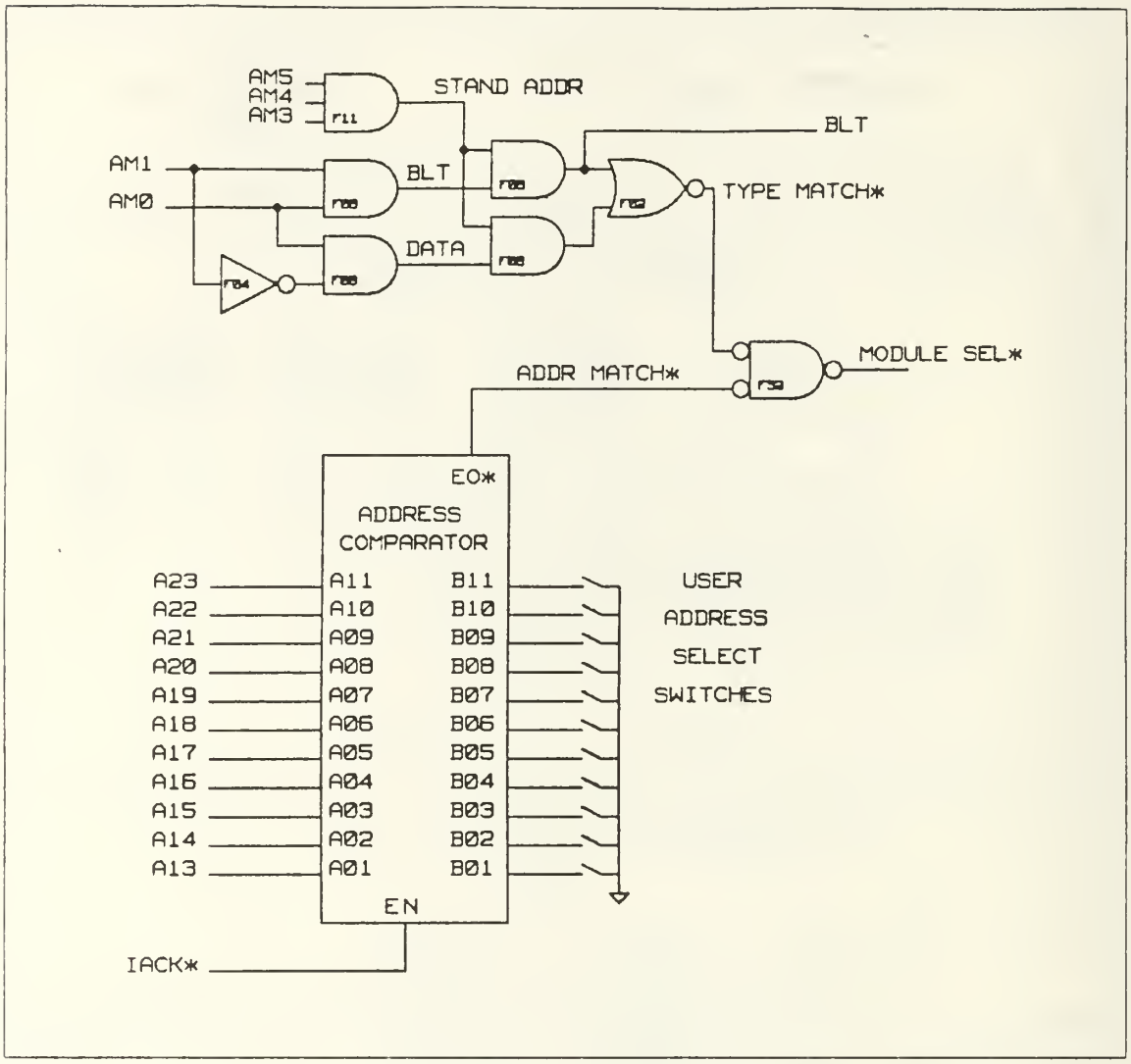


Figure 3.9 Module Select.

The module select circuit is disabled during interrupt acknowledge cycles (IACK* asserted). During interrupt acknowledge cycles only the lower three address lines, A01 thru A03, are valid. IACK* asserted disables the address comparator circuit and inhibits the assertion of MODULE SEL*. This ensures the DCM will ignore invalid address lines.

d. Bus Control

The bus control section, Figure 3.10, provides overall control to the host interface hardware. This section is based on the Signetics 68172 bus controller chip (BUSCON) which is a special purpose device designed to meet VME bus requirements.

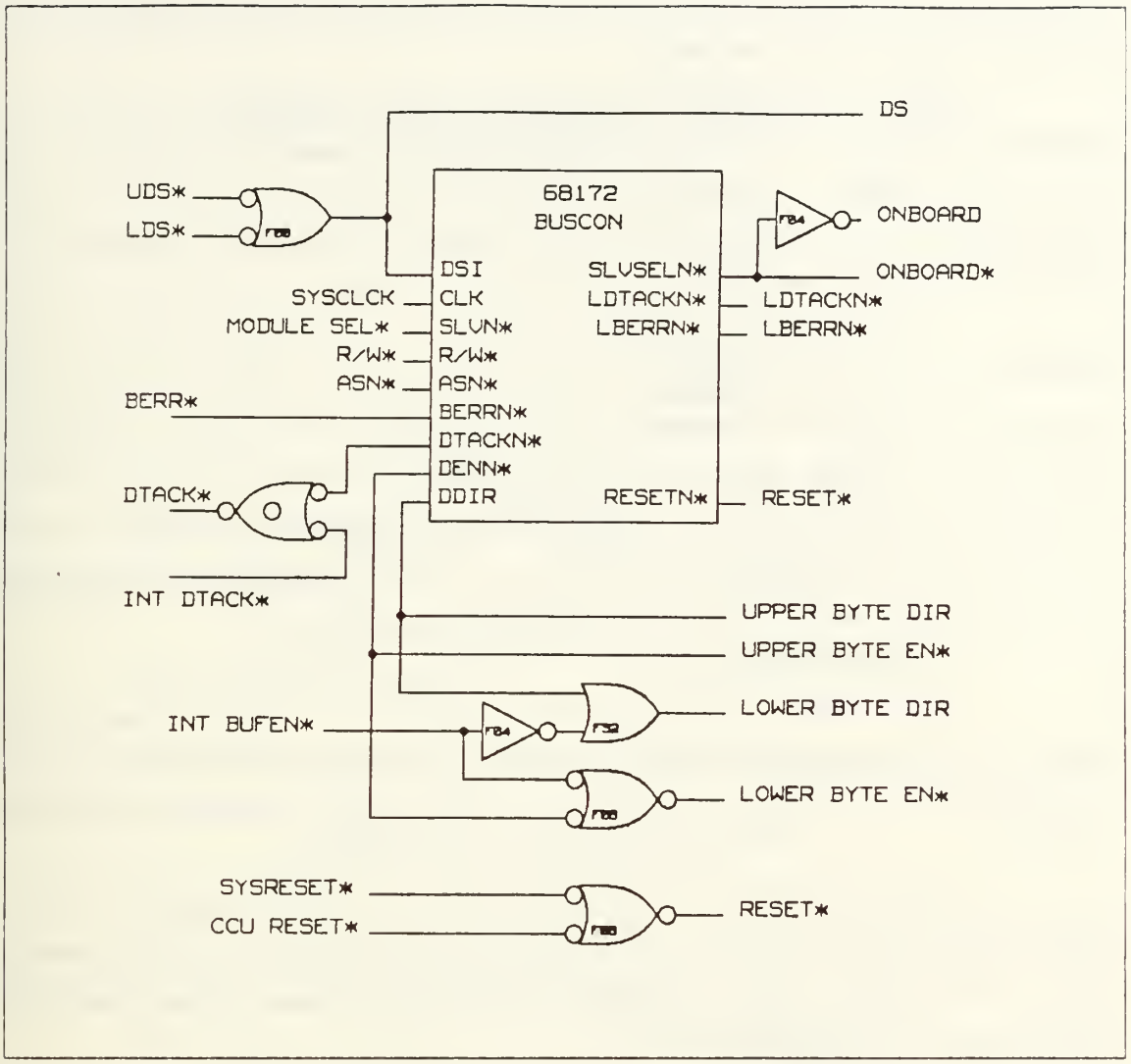


Figure 3.10 Bus Control.

The 68172 asserts the enable signal, ONBOARD, to the other interface circuits if the MODULE SEL* signal is valid for two trailing edges of the clock. This allows the address decoder circuits in the module select section sufficient time, 62.5 to 125 nanoseconds at 16 MHz, to properly evaluate the current memory cycle address. The ONBOARD signal enables all other circuits, except the VME bus generator, to perform their tasks. The VME bus generator operates only during interrupt cycles, which do not require DCM selection via normal addressing.

Data enable and direction signals are generated in this section and are shared between data transfer cycles and interrupt acknowledge cycles. The interrupt

acknowledge cycle uses only the lower byte of the data bus. This sharing allows only one set of transceivers to be used.

Bus error (BERR*) and data transfer acknowledge (DTACK*) are generated by the 68172 in response to signals from the timer in the read write control section. The BERR* and DTACK* can be connected directly to the VME bus. The DTACK* is shared with the VME bus interrupt data transfer acknowledge (INT DTACK*) by the wire ORing of the open collector outputs.

The reset signal (RESET*) is used to stop all host interface activity. RESET* puts all major components of the host interface in a clear or nonactive state, but the host interface remains ready to accept the next VME bus cycle. RESET* can be asserted by the host via the VME bus reset signal (SYSRESET*) or by the CCU via an internal reset (CCU RESET*). The SYSRESET* immediatly resets all DCM circuits and forces the DCM into a power-on reinitialization cycle. SYSRESET* is asserted by the host as a last resort to recover from a catastrophic failure.

e. Address Counter/Latch

The address counter latch section, Figure 3.11, latches address bits A01 thru A12 and increments the address during block transfer operations. This section provides address lines to the memory modules and read/write control for memory bank selection.

The circuit consists of a parallel loading 8 bit binary counter for A<8..1> and four D flip-flops for A<12..9>. The binary counter acts as a transparent latch when the load input (LD*) is asserted and latches the input address when LD* is negated, i.e. on the rising edge of LD*.

The LD* signal is driven low by ASN* going low at the beginning of the memory cycle and before ONBOARD* goes low. This is the state during A13 thru A23 decoding prior to DCM selection. When selection takes place, ONBOARD* will be asserted forcing LD* to go high, generating the rising edge needed to load the counter latches and D flip-flops. The LD* signal will not generate another rising edge until the DCM is deselected and reselected to begin a new memory cycle.

Block transfers are treated as a continuous memory cycle because ASN* remains asserted through the entire transfer. In this case the block transfer enable (BLT) with DCM selection (ONBOARD*) provides a count pulse to the counter when the lower data strobe (LDS*) is negated. This allows the address to increment between data strobes. This circuit requires that the data strobe be low before the DCM is

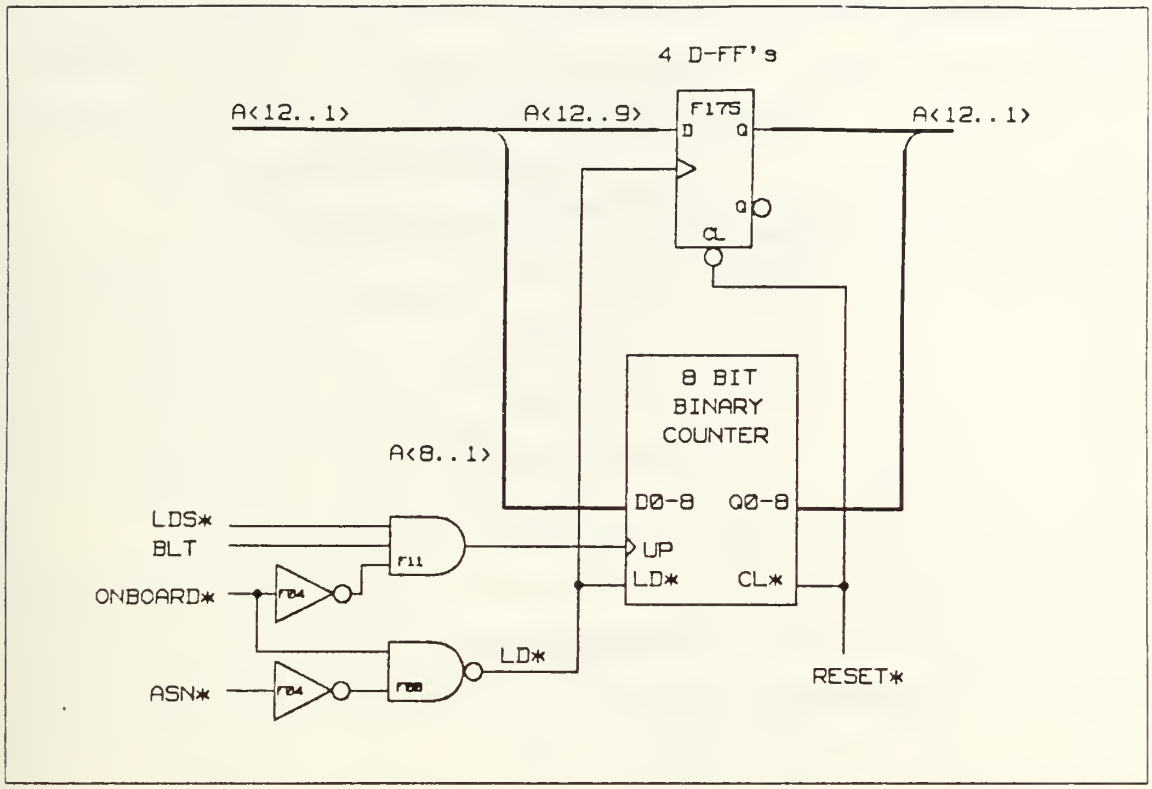


Figure 3.11 Address Counter/Latch.

selected. This is guaranteed by the VME bus specification which requires the data strobes be asserted within 10 nanoseconds of ASN* assertion, and the DCM selection delay during module select decoding, a minimum of 62.5 nanoseconds.

The counter requires a maximum of 27 nanoseconds to increment and present stable outputs. The data strobes will be negated, high, for a minimum of 30 nanoseconds. This ensures the address is incremented and stable by the start of the next data transfer.

f. Read/Write Control

The read/write control section, Figure 3.12, generates the signals required to access the dual ported memories and terminate individual transfers.

The read/write control signals are typical of most memory systems. The chip enable signals (DATA SEL* and CONT SEL*) select the proper memory bank based on A12; A12=0 selects control memory and A12=1 selects data memory. The chip enable signals enable both upper and lower bytes to be accessed in the selected memory bank. The read/write and output enable selects the upper and/or lower byte to be accessed.

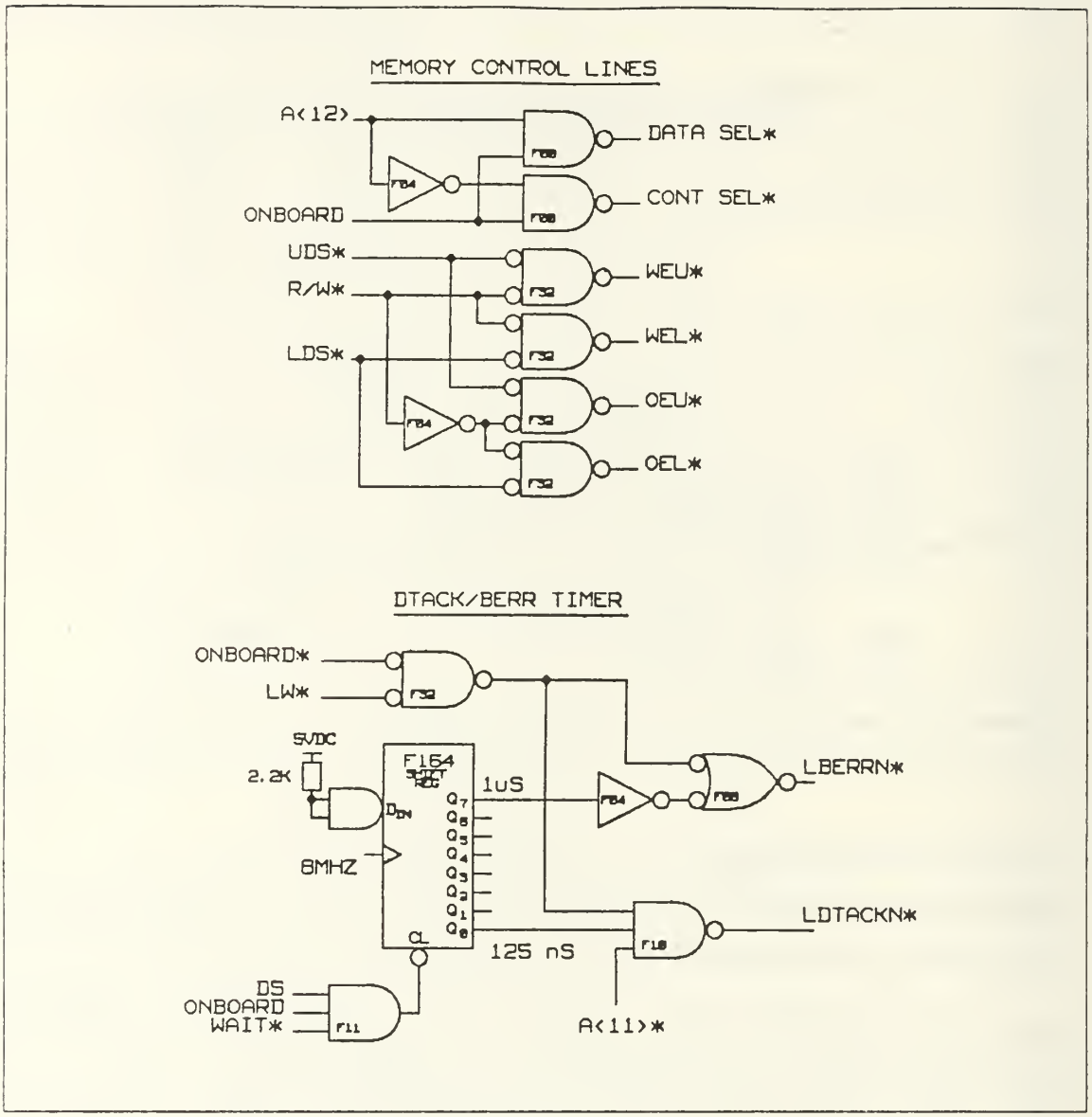


Figure 3.12 Read/Write Control.

Normal termination of individual transfers is accomplished by the data transfer acknowledge (LDTACK*). LDTACK* is generated if a valid address is selected (ONBOARD asserted and address bit A11 clear), if the addressed location is not busy (WAIT* not asserted), and if 125 nanoseconds has elapsed since the data strobe was asserted. A11 set selects the unpopulated portion of the DCM's memory. The assertion of WAIT* indicates the addressed location is busy. WAIT* will inhibit starting the timer until the WAIT* signal is negated. The DCM can not respond to 32 bit data transfers (LW* asserted) so long word transfers will inhibit LDTACK*.

Abnormal termination is accomplished with the bus error (LBERR*) signal. LBERR* is asserted one microsecond after data strobe assertion if no valid address is decoded or immediately if a long word transfer is attempted.

g. Command Interrupt Generator

The command interrupt generator section, Figure 3.13, generates interrupts to the CCU when the last byte in a disk command space is written to. These interrupt byte locations are (HEX):

- 108 for disk1
- 205 for disk2
- 302 for disk3
- 3FF for disk4

This circuit uses four D flip-flops as interrupt flags, one per disk, that generate the interrupt requests to the CCU. The D flip-flops have asynchronous preset and clear inputs. These inputs are used to provide timing isolation between the host interface and CCU. The flag flip-flops are cleared during the interrupt acknowledge cycle from the CCU.

The flag flip-flops are set by the command interrupt address decoder. This circuit decodes address lines A01 thru A12 for the above listed combinations and generates the flag set signals when they are written to.

h. VME Bus Interrupt Generator

The VME bus interrupt generator section, Figure 3.14, sends interrupts to the host via the VME bus. It can interrupt on any of the seven interrupt lines and provides an 8 bit vector over the bus during the interrupt acknowledge cycle. This circuit is based on the Signetics 68154 interrupt generator (INT GEN) chip which is a special purpose device designed for VME bus use.

The 68154 is a programmable device that can be set to interrupt on any interrupt level and more than one interrupt level may be used at the same time. The CCU can also program part of the vector sent during the interrupt acknowledge cycle. The upper seven bits of the vector are provided from a register within the 68154, the upper five of these are provided by the CCU and the remaining two reflect A2 and A3 from the VME bus. The lowest bit is provided externally by the designer and is normally A1. The lower three bits in the vector then represent the interrupt level being acknowledged.

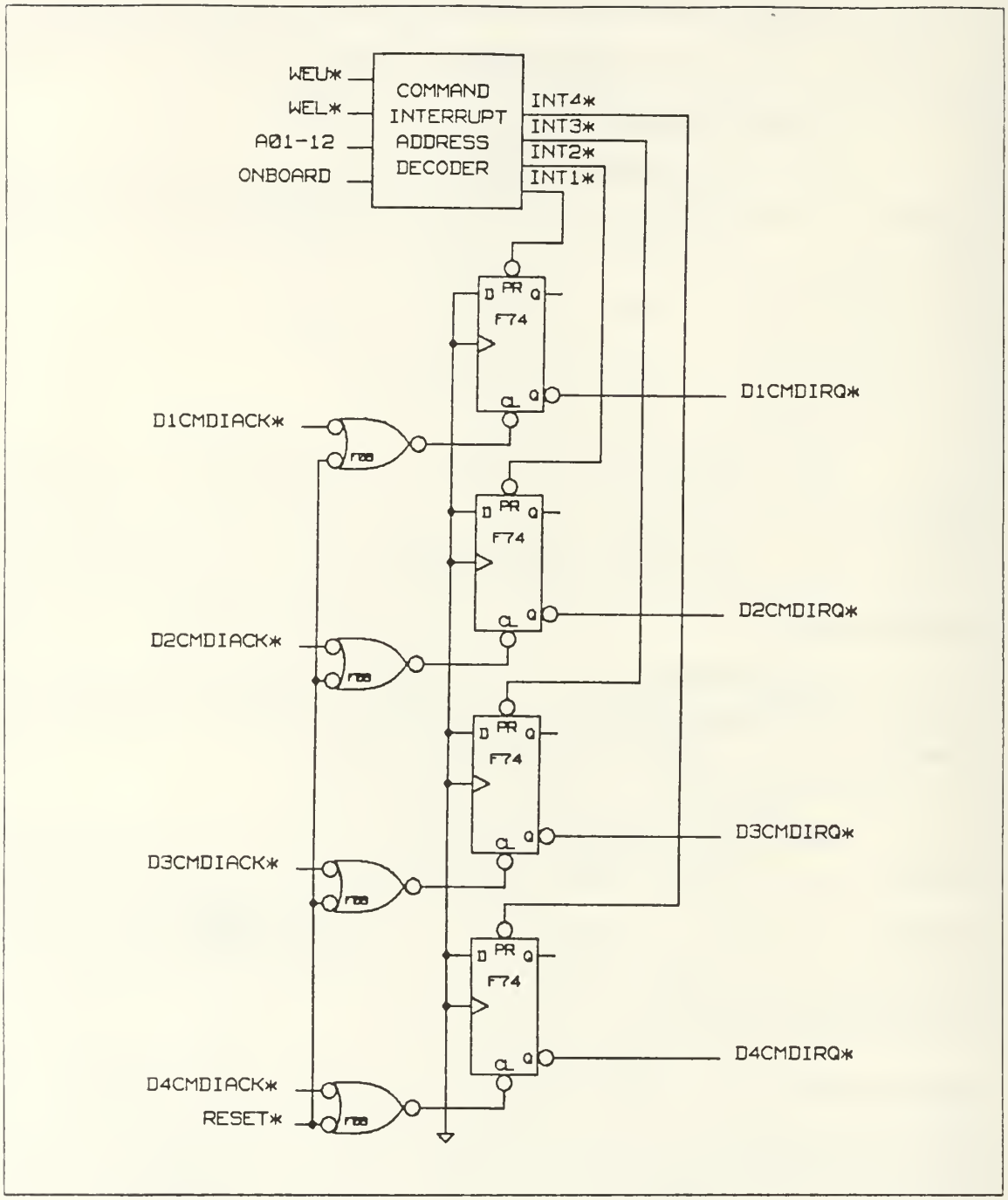


Figure 3.13 Command Interrupt Generator.

This arrangement allows the host to specify, as part of the command input, what level to interrupt on, and the upper five bits of the vector to be returned, when the command completion interrupt is generated. This is ideal for a multiuser system where the users may be on different interrupt levels.

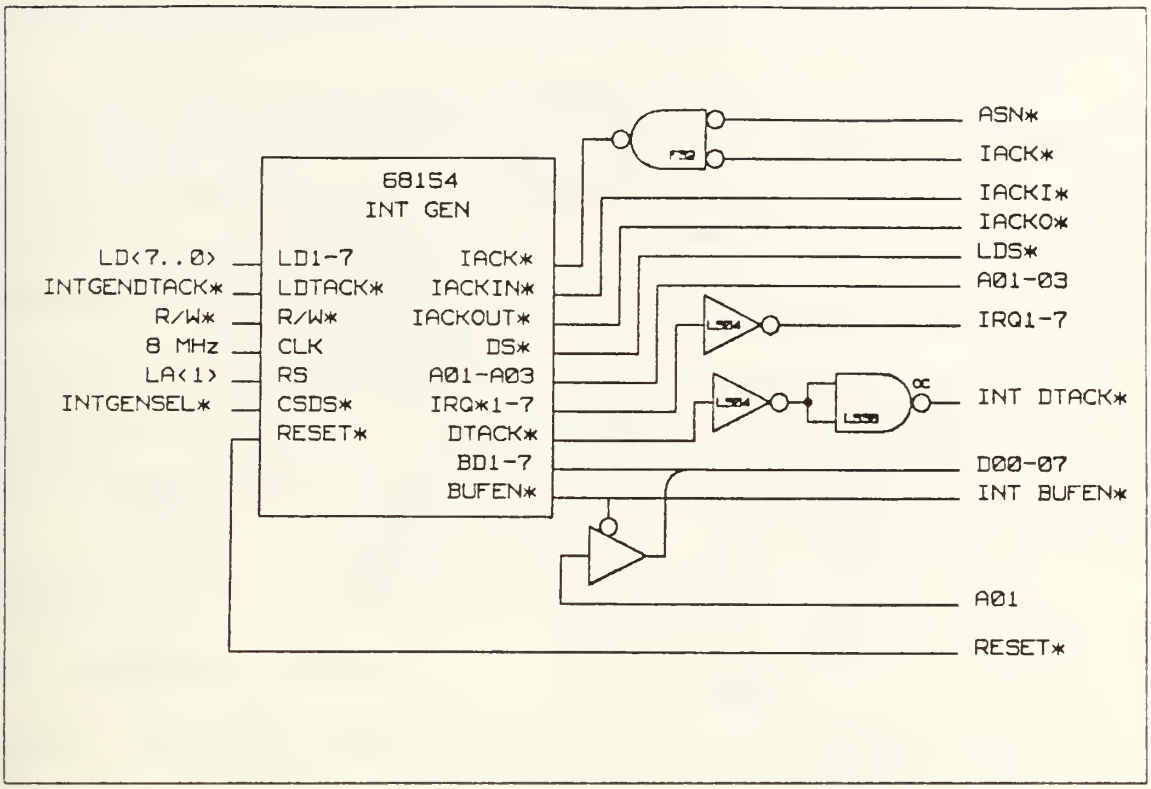


Figure 3.14 VME Bus Interrupt Generator.

The 68154 resides in the VME bus interrupt acknowledge daisy chain and will provide the interrupt vector if it has an interrupt active on the level currently being acknowledged. If it does not have one it passes the interrupt acknowledge in (IACKI*) down the daisy chain with IACKO*.

The interrupt data transfer acknowledge (INT DTACK*) and data transceivers for the vector are shared with the data transfer section as previously discussed.

The RESET* input resets all internal registers and negates any pending or active interrupt requests and IACKO*. The remaining inputs are from the CCU and will be discussed in the DCM control section.

C. DCM CONTROL

The DCM control section exercises control over all other sections. This is accomplished by the CCU, which programs the major devices in each section and initiates their actions. The DCM control section is based on a 68000 (CCU)

microprocessor system. The general arrangement of the DCM control section in relation to the other sections is given in Figure 3.15.

The 68000 treats all external devices as memory. The external devices such as memory, disk controllers, and interrupt generators are combined into the blocks labeled local devices and global devices in Figure 3.15. These devices share the same type control signals driven by the bus to which they are attached. The local devices are those that are accessed by the CCU on the local bus and the global devices are those that are shared by the CCU and DMAC on the global bus.

This arrangement was selected to allow simple addressing of all devices and to provide a common address space for global devices as seen by the CCU and DMAC. Since there are two buses that may be in use at the same time, there will be two separate bus control sections:

- local bus control - consists of local DTACK/BERR section and local memory control section
- global bus control - consists of global DTACK/BERR section and global memory control.

The bus control sections are independent to allow both busses to operate concurrently, but share common addresses for the global devices. The internal memory map, as seen by the CCU and DMAC, is provided in Figure 3.16. The CCU can access all devices on either bus but the DMAC can access only devices on the global bus. The lower thirteen address bits for the global devices are the same in the CCU and DMAC address spaces. The common address space for the global devices is required to allow the same address decoding when accessed by the CCU or DMAC.

1. **CCU Local Memory**

The CCU local memory devices are shown in Figure 3.17. The local memory consists of the host control buffer memory (port B) as previously discussed, a shared memory between the CCU and DMAC (DMACRAM port A) which is a dual ported memory like the host control buffer memory, eight kilobytes of ROM for internal control software, and eight kilobytes of RAM for CCU scratch-pad use. The ROM and RAM memories are not specified since any generic devices of the proper size will work. The only requirement is that the memories have access times less than 250 nanoseconds to allow zero wait state operation.

The DMACRAM dual ported memory is used to minimize bus contention when the continue or array modes of the DMAC are used. These modes allow multiple block transfers by putting the block sizes and limits in memory accessible by the

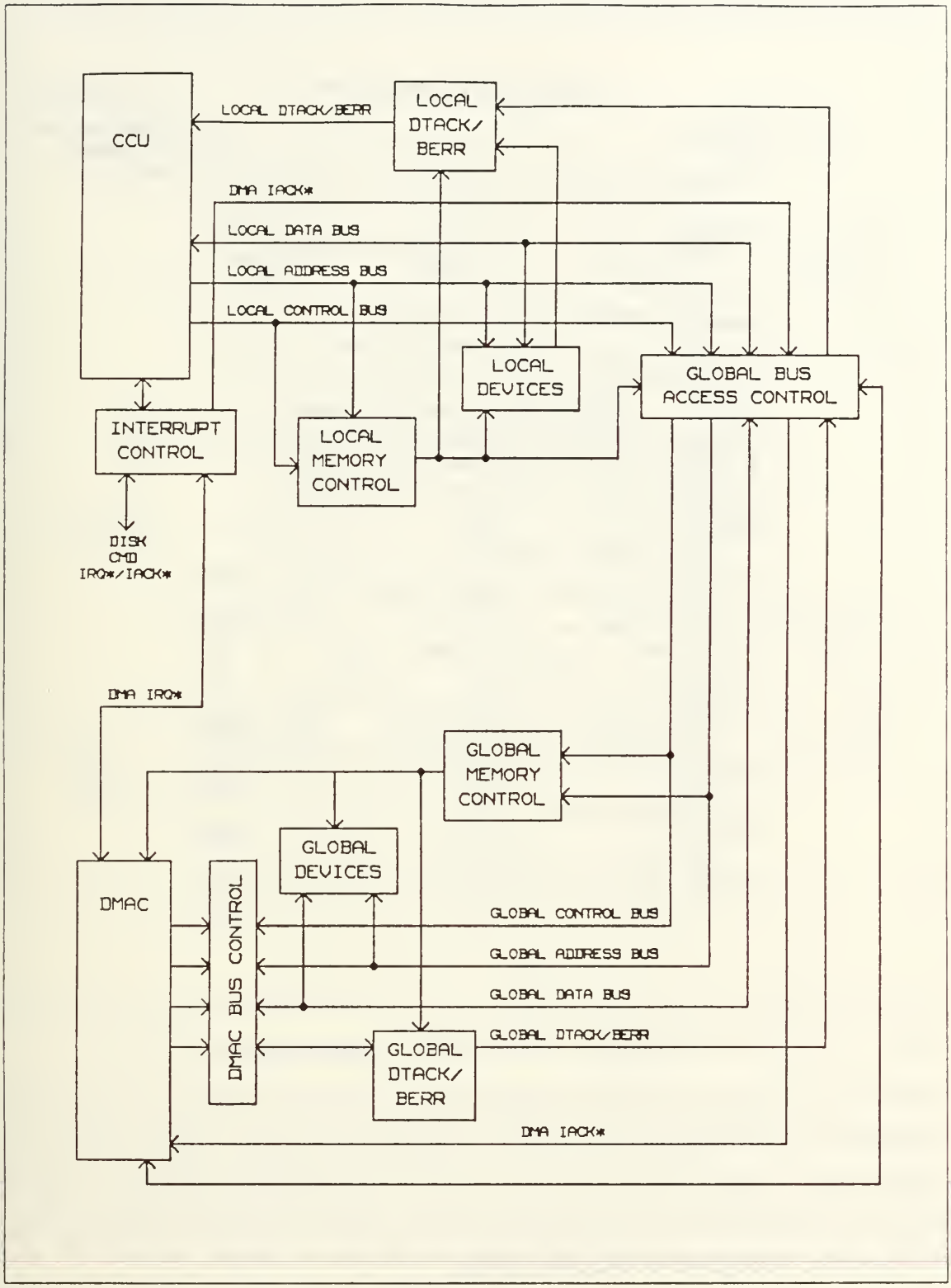


Figure 3.15 DCM Internal Organization.

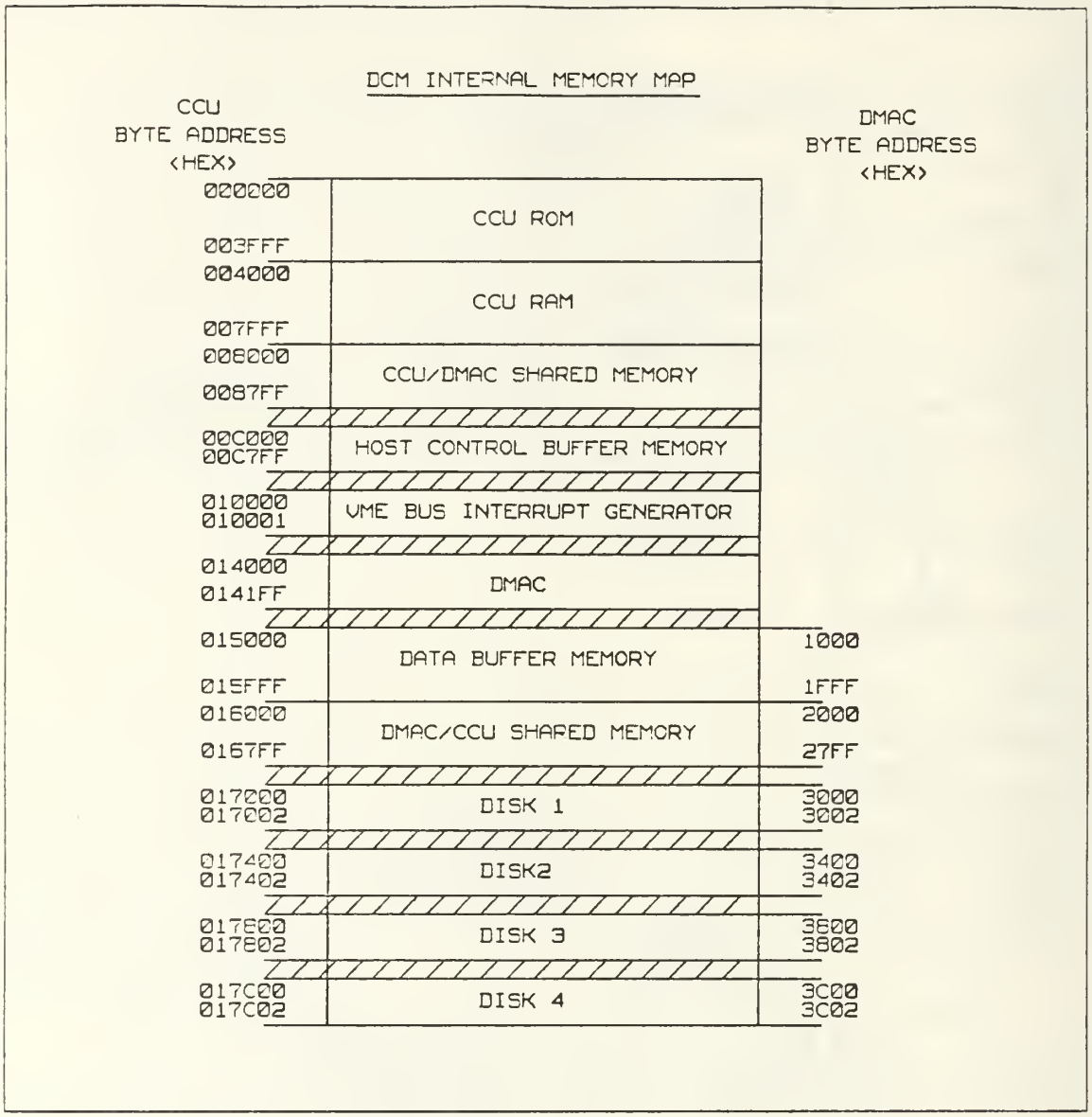


Figure 3.16 DCM Internal Memory Map.

DMAC. This allows the DMAC to move blocks of data without CCU intervention. The CCU can put the parameters into this memory via port A and the DMAC can retrieve them via port B with no bus contention. If single ported memory is used, the DMAC would need access to the local bus and contend with the CCU for access. This would reduce throughput as discussed earlier.

The VME bus interrupt generator (68154 INT GEN), CCU control side, is shown in Figure 3.18. It is treated as a two word memory block in local memory but

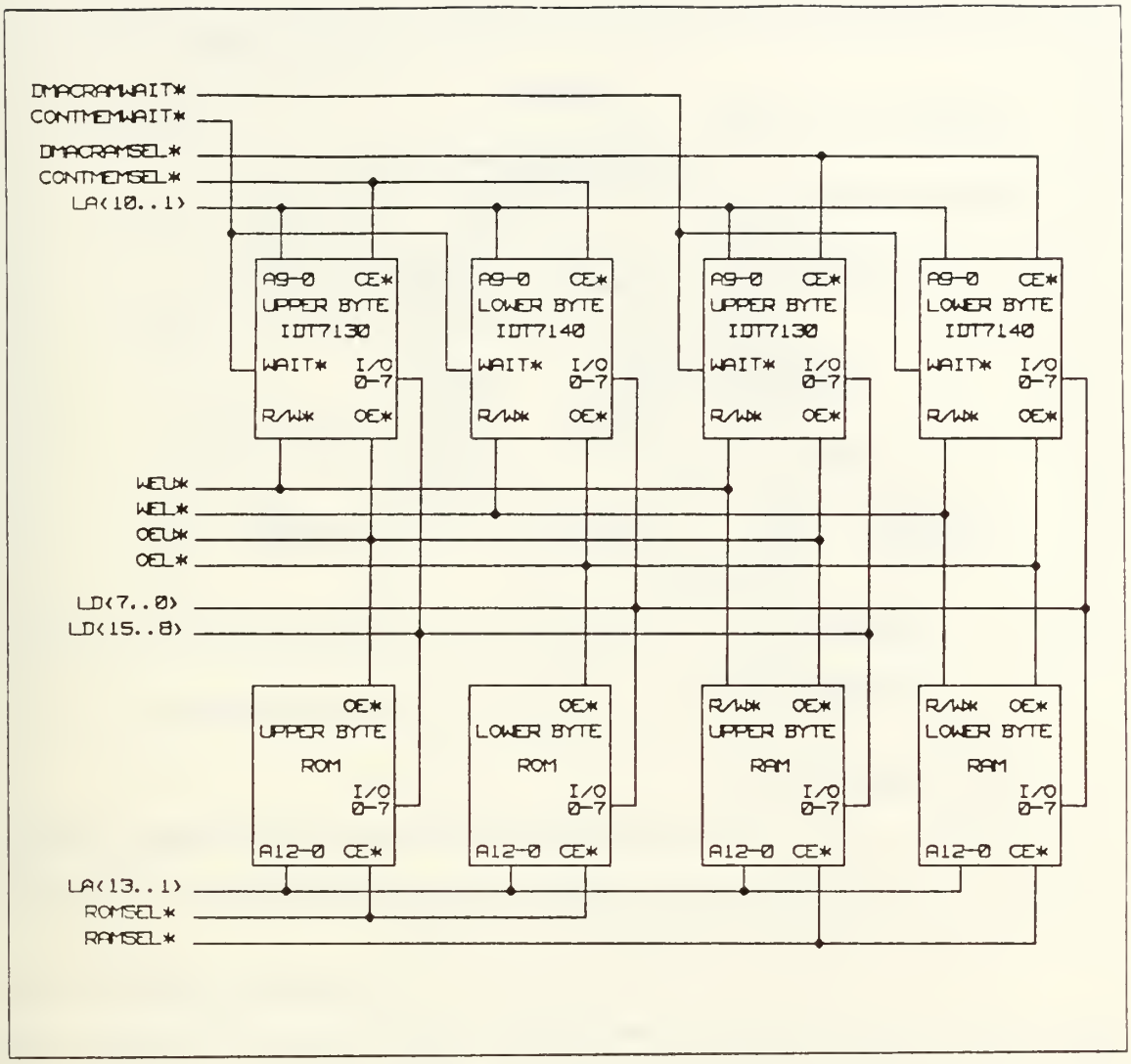


Figure 3.17 CCU Local Memory.

only the lower byte of each word is used. The 68154 has two internal registers, R0 and R1, that control interrupt operation. R0, LA<1> = 0, is the interrupt vector register which contains the upper five bits of the interrupt vector to be provided to the host via the VME bus during interrupt acknowledge cycle. The lower two bits of R0 are used to enable all interrupts and reset all interrupt levels. R1, LA<1> = 1, is the interrupt request register, setting bit n in this register generates a level n interrupt on the VME bus. The interrupt request bit will be reset when the host acknowledges that interrupt level.

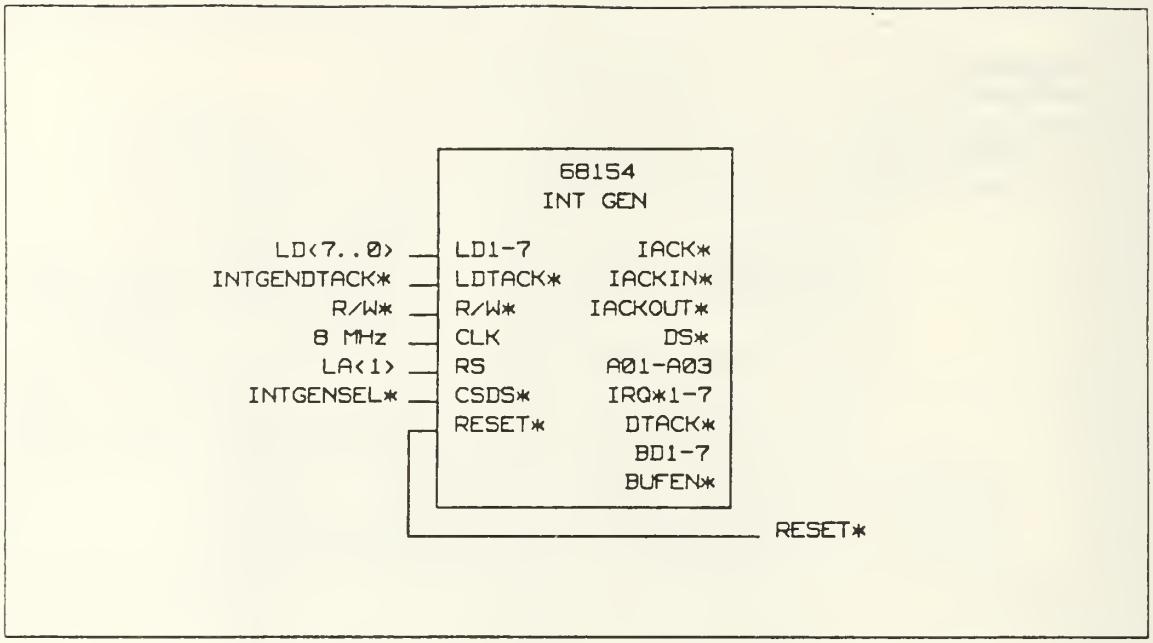


Figure 3.18 CCU Control of VME Bus Interrupt Generator.

2. CCU Support Circuits

Figure 3.19 shows two support circuits not shown in prior diagrams. These are the clock generator circuit, which provides all internal clock signals, and the reset generator, which provides the internal CCU reset signal.

The clock generator circuit is a simple binary counter that divides the 16 MHz input by 2, 4, 8 and 16 to produce the indicated outputs. The 16 MHz, 8 MHz, and 4 MHz outputs are used as the timing signals for the other sections.

The reset circuit can be activated in four ways:

- SYSRESET* from the VME bus directly generates the internal CCU reset
- manual reset from a switch to reset the DCM without affecting the host
- power on reset to initialize the DCM when power is first applied
- software reset initiated by the 68000 reset instruction to reinitialize all DCM circuits except the 68000

The SYSRESET* and manual reset are provided to recover from a catastrophic failure and are used as a last resort. On reset the DCM reinitializes itself as if power was first applied and all data and commands are lost. The power on reset is a simple timer to provide a 100 millisecond reset pulse after power is first applied to initialize the DCM for initial operation. The software reset provides a means for the internal control software to initialize the hardware without resetting the 68000.

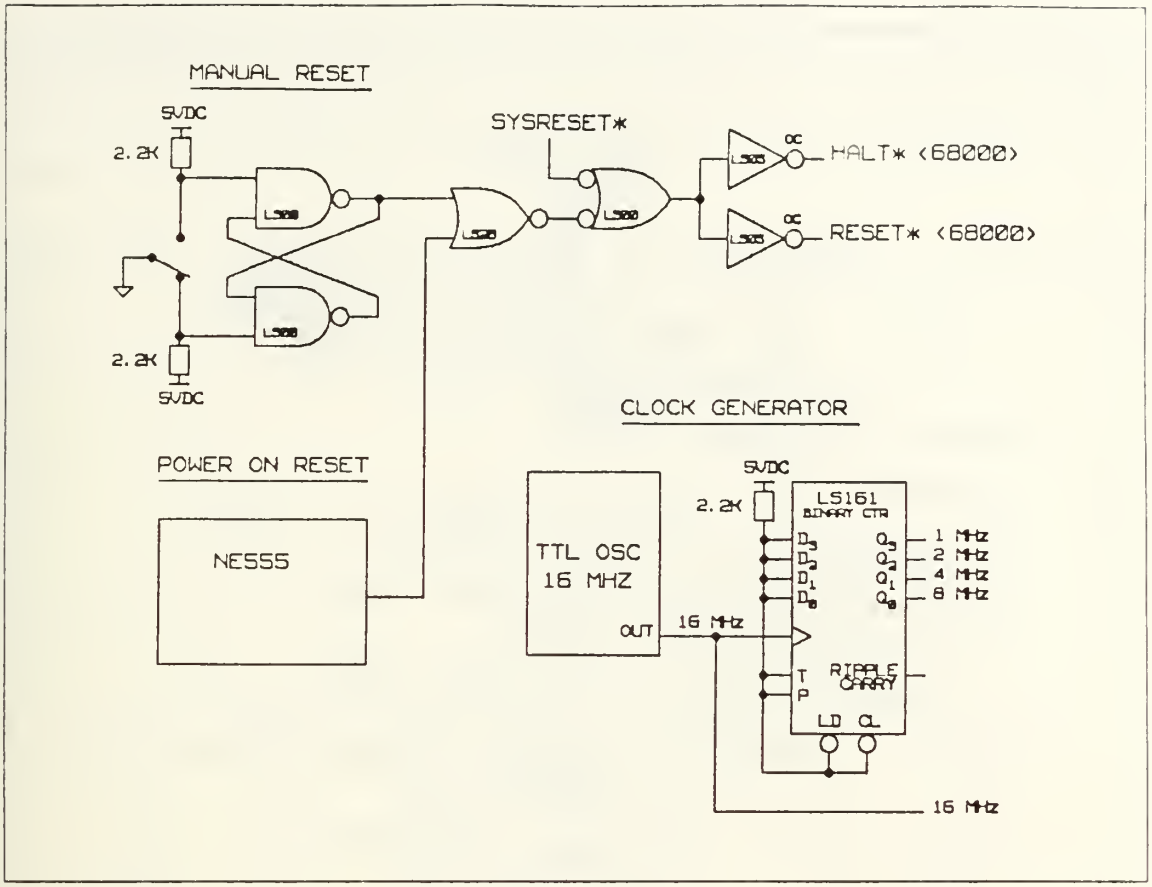


Figure 3.19 Clock Generator and Reset Circuits.

3. CCU Bus Control

The CCU bus control section generates the signals required for the CCU to access all devices on the local bus and initiate requests for the global bus. The control signals are typical memory controls since all devices are treated as memory by the 68000. The CCU bus control section consists of a local memory control section and a local DTACK/BERR generator section. The memory control circuits are shown in Figure 3.20.

The memory select circuit decodes local address bits 14 thru 16 to generate the device select enables. The device enables are used to enable the individual devices for read or write operations with the CCU. The global bus request (GBUSREQ*) is also generated here when the address bits are 101 respectively, this is the address space of the global bus devices in the CCU memory map.

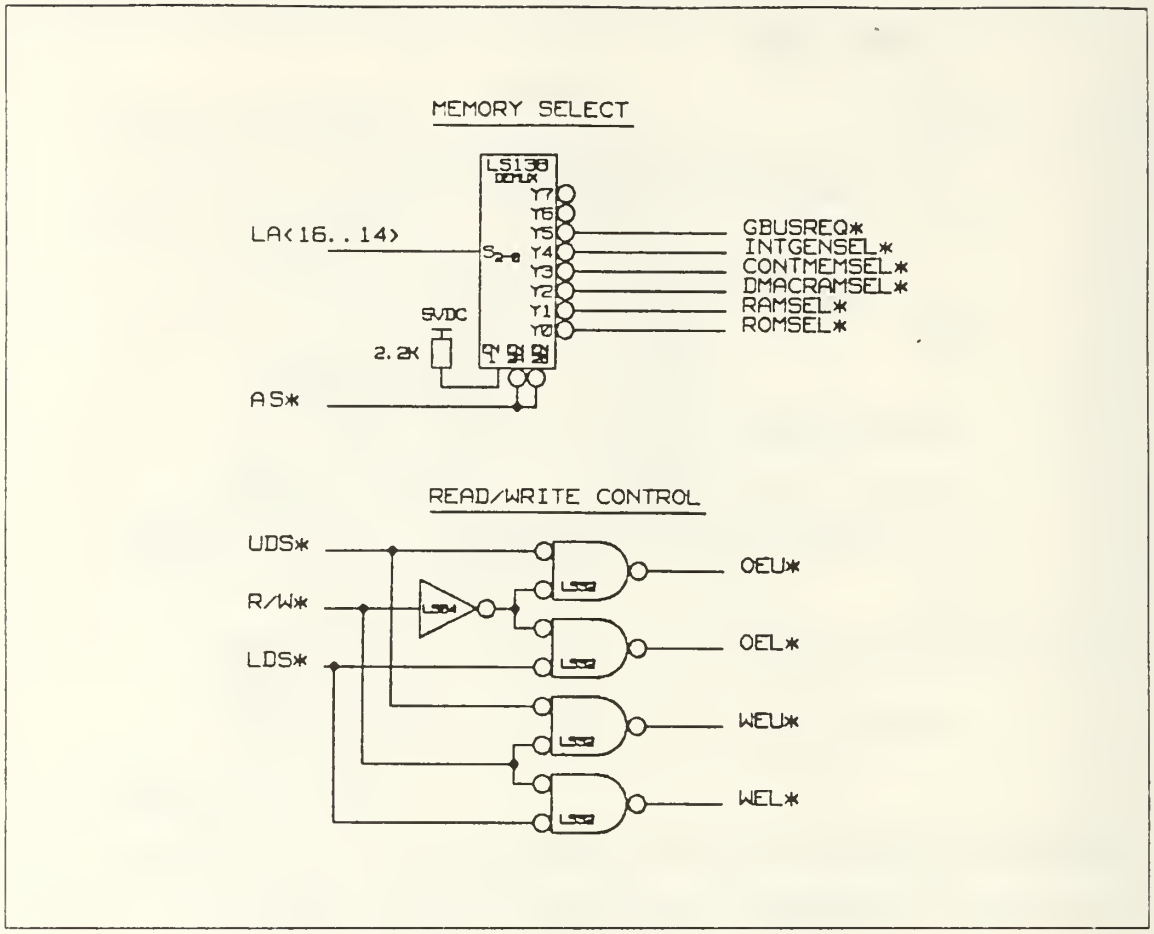


Figure 3.20 Memory Control Circuits.

The read/write control circuit generates the output enables (OEU* and OEL*) for read operations and write enables (WEU* and WEL*) for write operations. The output and write enables are asserted on a byte basis to allow byte and word operations with the selected device on the local bus.

For global bus devices, the GBUSREQ* is a request to the global bus access controller for global bus access. The memory control signals mentioned above do not take part in accesses to global bus devices. The global bus control section provides the control signals required for global devices and functions the same as the local bus controller.

The local DTACK/BERR generator is shown in Figure 3.21. This circuit terminates local data bus transfers (LDTACK*) and generates the local bus error (LBERR*) signal if an erroneous address is referenced. This circuit uses a shift register

as a timing device to provide the access time required for the memory devices, 250 nanoseconds, and a 2 microsecond time out to indicate that an unpopulated part of memory was referenced. The timing signals are combined with the device selects to allow the use of devices with various access times.

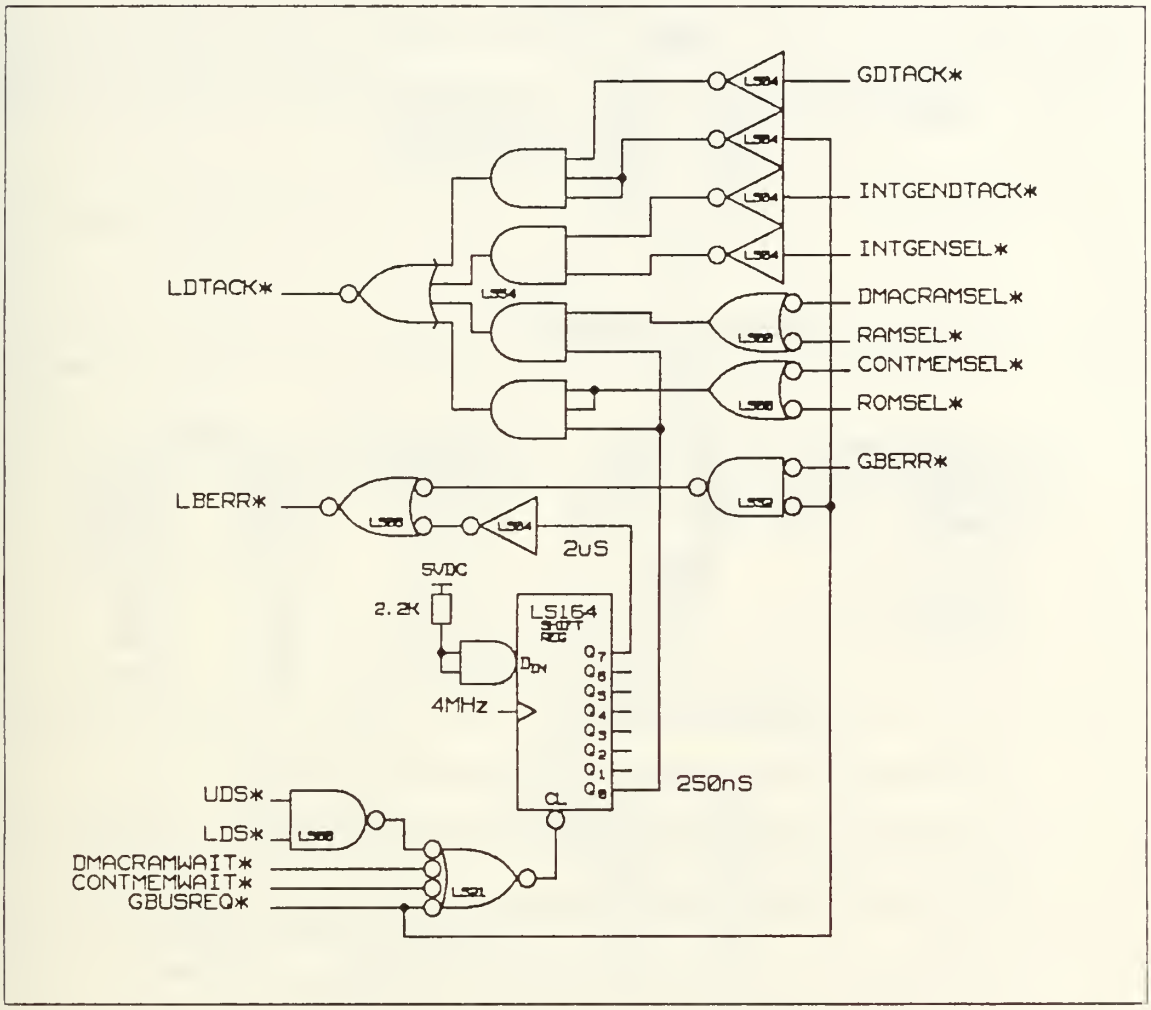


Figure 3.21 Local DTACK, BERR Circuits.

During CCU accesses to the global bus, the CCU and DMAC may be contending for control of the bus with no fixed time for the CCU to gain control. This means the timer in Figure 3.21 can not be used to terminate the transfer. In this case, the CCU waits by asserting GBUSREQ* which disables the timer and enables the global data transfer acknowledge (GDTACK*) and global bus error (GBUSERR*) inputs to the LDTACK* and LBERR* circuits. The GDTACK* and GBERR* can not

be asserted on the local bus until the CCU has control of the global bus. The GDTACK* or GBERR* will then terminate the transfer after the CCU has control of the bus and the selected devices have had enough time to respond.

4. **Interrupt Control**

The interrupt control section prioritizes interrupt requests, generates interrupt acknowledges and provides interrupt vectors for those devices that do not provide their own vectors. A summary of the interrupt codes used is provided in Table 1 and Figure 3.22 shows the circuits that generate these codes.

TABLE 1
INTERRUPT CODES

INTERRUPT		ENCODED	ACKNOWLEDGE ADDRESS	VECTOR NUMBER	VECTOR ADDRESS
SOURCE	LEVEL	IPL<2..0>*	LA<3..1>	<DEC>	<HEX>
DMACIRQ*	5	0 1 0	1 0 1	DMAC WILL PROVIDE	
D4CMDIRQ*	4	0 1 1	1 0 0	68	110
D3CMDIRQ*	3	1 0 0	0 1 1	67	10C
D2CMDIRQ*	2	1 0 1	0 1 0	66	108
D1CMDIRQ*	1	1 1 0	0 0 1	65	104

There are two types of interrupts provided for in the DCM:

- command present interrupts (D1CMDIRQ*-D4CMDIRQ*) generated in the host interface as previously described
- DMAC interrupts (DMACIRQ*) which include interrupts generated by the disk controllers

The DMAC is configured by the CCU to provide interrupts and vectors on completion of DMAC operations or in response to disk controller interrupt requests. The CCU configures the DMAC, via software, by writing interrupt vectors into the interrupt vector registers in the DMAC for each of the attached disk controllers and by enabling disk controller interrupts. This allows the DMAC to consolidate the four disk controllers interrupt request lines into a single interrupt request line and provide a unique vector for each disk controller.

Interrupt requests to the CCU are prioritized by the interrupt priority encoder with DMAC interrupts as the highest priority. DMAC interrupts are the highest

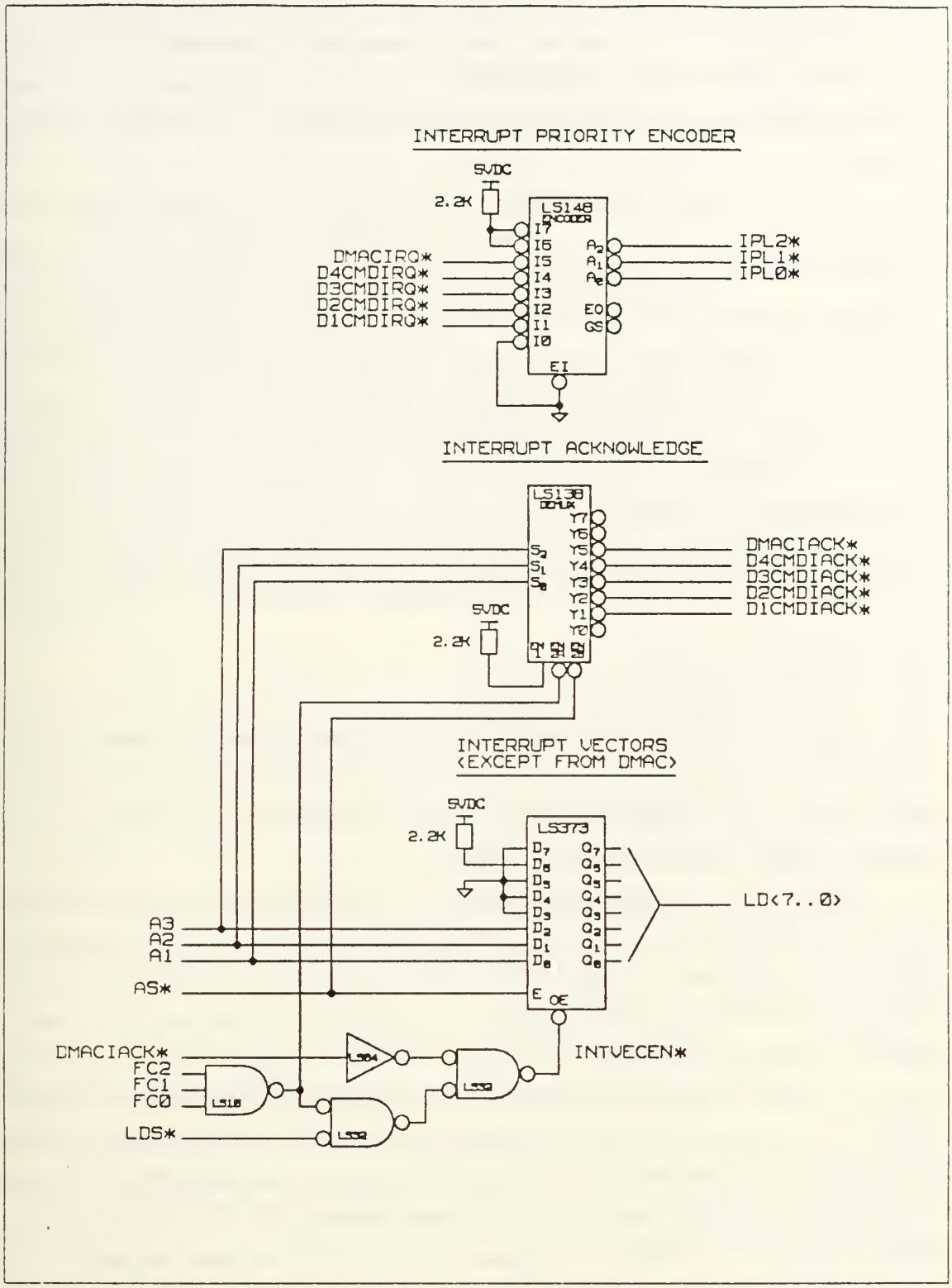


Figure 3.22 Interrupt Control Circuits.

priority because they indicate a data transfer termination and will release assets needed for a new command. The command present interrupts are prioritized in the hardware to allow easy differentiation between them during interrupt acknowledge cycles. The interrupt handling software treats the command present interrupts equally, first-come, first-served.

The interrupt acknowledge circuit decodes the interrupt level being acknowledged, LA<3..1>, to generate the individual interrupt acknowledges required to put the proper vector onto the data bus. The command present interrupt acknowledges (D1CMDIACK*-D4CMDIACK*) to the CMD present flip-flops reset the flip-flops generating the interrupts to ensure that a command is recognized only once. The DMAC interrupt acknowledge (DMACIACK*) enables the DMAC to provide the interrupting disk controller's vector and inhibits the interrupt vector circuit.

The interrupt vector circuit is used to provide the interrupt vectors for the command present interrupts. The upper five bits of the vector are fixed to 01000 and the lower three bits differentiate between the individual command present interrupts. This circuit is disabled during DMAC acknowledge cycles because the DMAC provides the vector.

D. GLOBAL BUS ACCESS CONTROL

The global bus access control section coordinates the orderly access of the CCU and DMAC to the global bus. The CCU and DMAC function as global bus MASTERs, driving the global bus address and control lines and controlling data line direction, when they have control of the bus.

Global bus access follows the 68000 bus arbitration protocol. The 68000 bus arbitration protocol is a three signal handshake (request-grant-acknowledge) protocol that ensures only one bus MASTER is given bus control at a time. A MASTER requests bus access by asserting a bus request (BR*) to the bus controller (68000 or external control unit). The bus controller asserts bus grant (BG*) to the highest priority requester which indicates that the requesting MASTER may take control of the bus when the current MASTER relenquishes the bus. The requesting MASTER then monitors the address stable (AS*) and bus grant acknowledge (BGACK*) signals to determine when the current MASTER relenquishes the bus. AS* negated indicates that the current bus cycle is completed and BGACK* negated indicates that the current bus MASTER has released the bus. After AS* and BGACK* are negated by the current bus MASTER, the requesting MASTER asserts BGACK* and becomes the

new bus MASTER. After asserting BGACK*, the new bus MASTER negates its BR* which causes the bus controller to negate BG* and start a new round of arbitration. A new round of arbitration can not begin until the new MASTER's bus grant is negated and bus mastership can not change while bus grant acknowledge is asserted.

The DCM's global bus access is controlled by two circuits:

- global bus arbitration circuit which determines which unit, CCU or DMAC, will be the global bus MASTER
- local global bus interface which connects the CCU local bus to the global bus when the CCU gains access to the global bus

1. **Global Bus Arbitration**

The global bus arbitration circuit is shown in Figure 3.23. The arbitration is prioritized with the DMAC global bus request (DMACGBR*) as the highest priority and the CCU global bus requests (CCUGBR*) are recognized for single cycle transfers only. Making DMAC global bus requests the highest priority and limiting the CCU to single cycle global bus transfers ensures that the DMAC can gain global bus control in time to service data transfer requests from the disk controllers without disk data overrun.

Global bus arbitration is performed by the Motorola 68452 bus arbitration module (BAM) which can prioritize up to eight potential bus MASTERs and follows the 68000 bus arbitration protocol described above. The BAM can be configured to operate in a local bus arbitration mode or a global bus arbitration mode.

In the local bus arbitration mode the BAM is used to prioritize bus requests for a 68000's local bus by passing the highest priority bus request to the 68000's bus request input and returning the 68000's bus grant output to the highest priority requester. The bus grant acknowledge signal from potential bus MASTERs are shared by the 68000, BAM, and all bus MASTERs to complete the bus arbitration handshake.

In the global bus arbitration mode the BAM serves as the central bus controller with no need to request the bus from a 68000. In this mode the highest priority bus request is passed directly from the BAM's bus request output (BR*) to the BAM's bus grant input (BG*) after a 50 nanosecond arbitration delay. The arbitration delay is required because there may be spiking on the individual bus grant output signals (DBG0*-DBG7*) during arbitration if multiple bus requests arrive at the same time. The spiking and arbitration will be resolved within 50 nanoseconds so delaying the BAM bus grant input will ensure that the individual bus grant output signals will not be asserted until arbitration is complete.

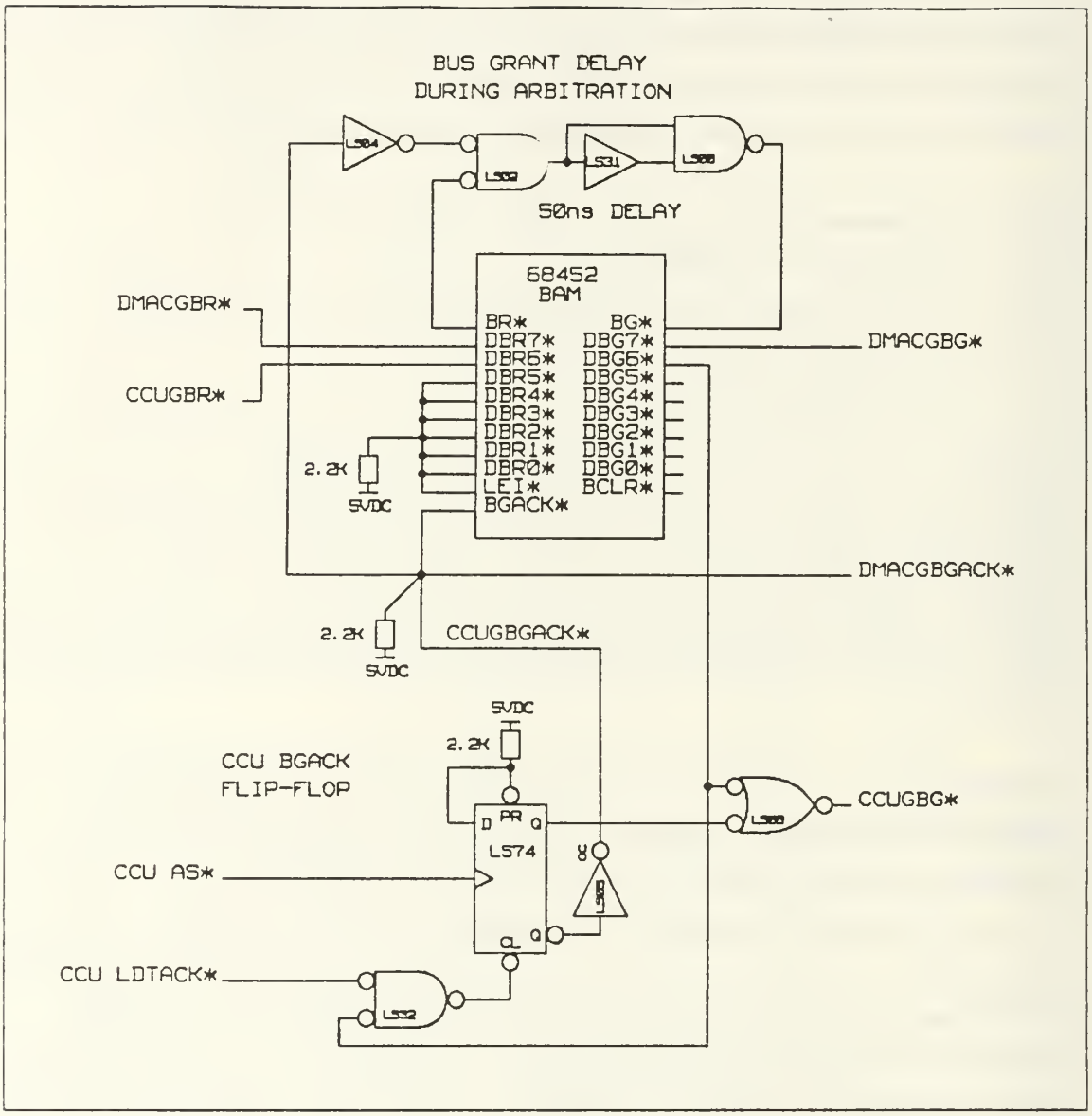


Figure 3.23 Global Bus Arbitration Circuit.

In the DCM, the BAM is configured for the global bus arbitration mode and acts as the central bus controller for the DCM's global bus. The DMAC global bus request is given top priority, level 7, and the CCU global bus request is given priority level 6. The remaining priority levels are not used. DMAC global bus access is handled in a straightforward manner in accordance with the previously described 68000 bus arbitration protocol because the DMAC has onchip bus control circuits that follow the 68000 protocol and allow the DMAC to function in a multiple bus MASTER

environment. CCU global bus access requires additional support circuits because the 68000 is designed to be the central bus controller, relenquishing the bus when required, and does not directly generate the bus request and bus grant acknowledge signals required for requesting the bus from another bus controller and holding the bus in a multiple bus MASTER environment. The additional support circuits for handling CCU global bus access are shown in the lower part of Figure 3.23.

CCU global bus requests (CCUGBR*) are generated in the local memory control section, as previously described in Figure 3.20, when the CCU addresses devices on the global bus. The BAM receives the request and will assert DBG6* in response if the DMAC is not using or requesting the global bus. Asserting DBG6* will generate the enable (CCUGBG*) to the local global bus interface that connects the local bus to the global bus and allows the CCU to drive the address, data, and control lines of the global bus. DBG6* assertion also enables the clear input of the CCU BGACK flip-flop.

The CCU BGACK flip-flop is used to generate the CCU global bus grant acknowledge (CCUGBGACK*) and limits CCU accesses to the global bus to single cycle transfers. In the normal state, when the CCU is not using the global bus, the CCU BGACK flip-flop is set, negating CCUGBGACK* and allowing normal operation of DMAC global bus transfers. When the BAM asserts DBG6*, indicating that the CCU is the current global bus MASTER, the clear input to the CCU BGACK flip-flop is enabled. Clearing the CCU BGACK flip-flop indicates that the CCU global bus transfer is terminating and as a result CCUGBGACK* is generated. The CCU BGACK flip-flop will be cleared when the addressed device asserts data transfer acknowledge which generates CCU LDTACK*. Asserting CCUGBGACK* causes the BAM to negate DBG6* but the CCUGBG* will remain asserted until the CCU global bus transfer is completed. The final step in a CCU global bus transfer is the negation of CCU AS*, which indicates that the transfer has completed. Negating CCU AS* sets the CCU BGACK flip-flop which negates CCUGBG* and CCUGBGACK*, completing the handshake and enabling a new round of arbitration.

2. **Local/Global Bus Interface**

The local global bus interface circuit, shown in Figure 3.24, forms a connection between the local bus and the global bus when the CCU has control of the global bus. In this case the address, data, and control signals are allowed to pass between the two buses.

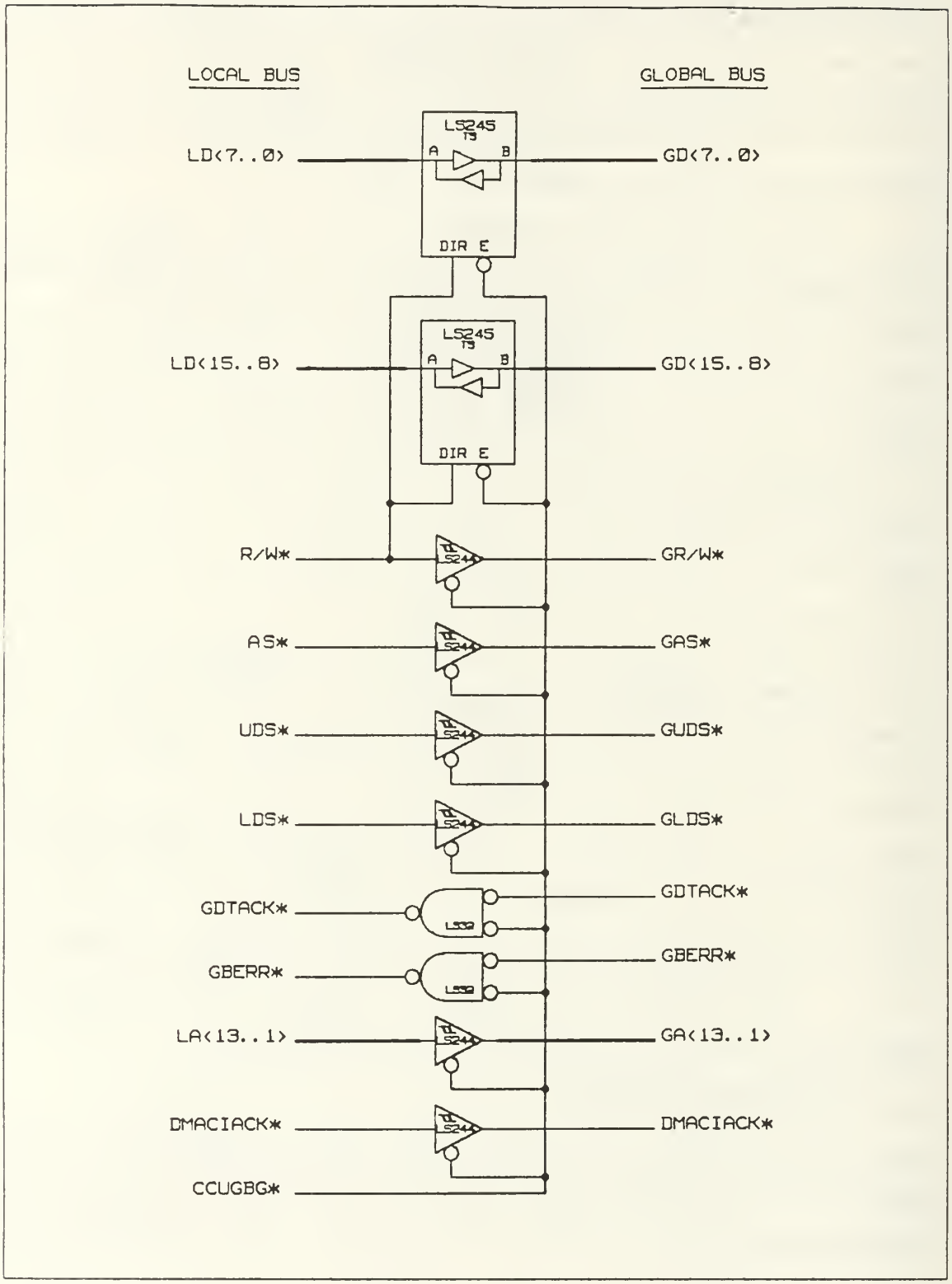


Figure 3.24 Local/Global Bus Interface Circuit.

When the CCU is the global bus MASTER, CCUGBG* is asserted and the CCU is enabled to drive both the global bus address lines (GA<13..1>) and global bus memory control lines (GR W*, GAS*, GUDS*, GLDS*). In addition the CCU can drive or receive the global bus data lines (GD<15..8>, GD<7..0>). The global bus data transfer acknowledge (GDTACK*) and global bus error (GBERR*) are enabled by CCUGBG* in order to be passed to the CCU to terminate the CCU global bus transfer.

The DMAC interrupt acknowledge signal from the CCU (DMACIACK*) to the DMAC passes through the local global bus interface to ensure that the DMAC does not receive the DMACIACK* signal while the DMAC is the global bus MASTER. If the DMAC were to receive the DMACIACK* signal while in the MASTER mode, the DMAC would terminate its current bus cycle, before completion, with an error and attempt to respond to the interrupt acknowledge with an interrupt vector. Gating the DMACIACK* signal through the local global bus interface ensures the DMAC is not in the MASTER mode when DMACIACK* is asserted because the DMACIACK* signal will not be put on the global bus unless the CCU is the global bus MASTER.

E. DMA CONTROL

The DMA control section controls data transfers between the disk controllers and data buffer memory. These transfers are via direct memory access (DMA) under the control of a Motorola 68450 direct memory access controller (DMAC). The 68450 can accept up to four DMA devices and has a built in bus controller to allow it to assume bus mastership when controlling the data transfers between the DMA devices and memory. Once the DMAC is programmed by the CCU, the data transfers will take place without further CCU intervention.

The DMAC is programmed by writing the transfer parameters into a set of internal control registers, 17 registers per attached DMA device plus one general control register. The DCM's internal control registers select the mode of transfer, addressing to be used, priority level, and interrupt vectors to be returned on transfer completion.

In the DCM, the DMAC is configured for the cycle-steal mode with implicit addressing. The cycle-steal mode causes the DMAC to relenquish the global bus if no data transfer requests are present from the disk controllers. This creates gaps in the data transfers so the CCU can gain access to the global bus to initiate transfers with

idle disk controllers. Implicit addressing is used to allow single cycle operand transfer operation, when a single read or write cycle can move the operand from source to destination. In implicit addressing, the DMAC provides a single address, the address of the memory location in the data buffer memory that is the source or destination of the operand. The DMA devices, disk controllers in this case, are not directly addressed but are controlled by the DMA request and acknowledge lines. In explicit addressing, the DMAC would address the operand source, read the source operand into a holding register, address the destination, and write the operand into the destination. This requires two memory cycles, a read cycle followed by a write cycle.

Figure 3.25 shows the signals used by the DMAC to control the global bus and DMA devices. The DMA device control signals allow the DMAC to control data transfers with DMA devices without explicitly addressing them as in a normal memory reference cycle. The data address demultiplex control signals are used to control the source of global bus signals when switching between the CCU and DMAC as a global bus MASTER. The DMAC global bus request, grant, and acknowledge signals follow the 68000 bus arbitration protocol previously discussed. The DMAC interrupt request (DMACIRQ*) and interrupt acknowledge (DMACIACK*) signals follow the standard 68000 vectored interrupt protocol.

The DMAC operates in two modes with the direction of the global bus control lines determined by the mode of operation:

- MPU mode - the state that the DMAC enters when the chip is selected (DMACSEL*) by the CCU in which the DMAC's internal registers are read or written by the CCU to initiate a data transfer or check status
- DMA mode - the state that the DMAC enters when acting as the global bus MASTER to perform an operand transfer with the disk controllers

In the MPU mode, the DMAC acts like a CCU slave device with the CCU reading or writting the DMAC internal registers. In this mode the global address lines, GA<7..1>, are inputs that select the DMAC internal register to be operated on and the global data lines, DMAC<A8, D0-A23, D15>, are inputs or outputs for the operand transfer. The memory control lines (GAS*, GR W*, GUDS*, GLDS*, DMACSEL*) are inputs used in the normal 68000 memory access protocol. The DMAC data transfer acknowledge (DMACDTACK*) is an output to indicate to the CCU that the transfer is complete.

In the DMA mode, the above signal directions are reversed because the DMAC is providing address and control signals as the global bus MASTER. In this mode the

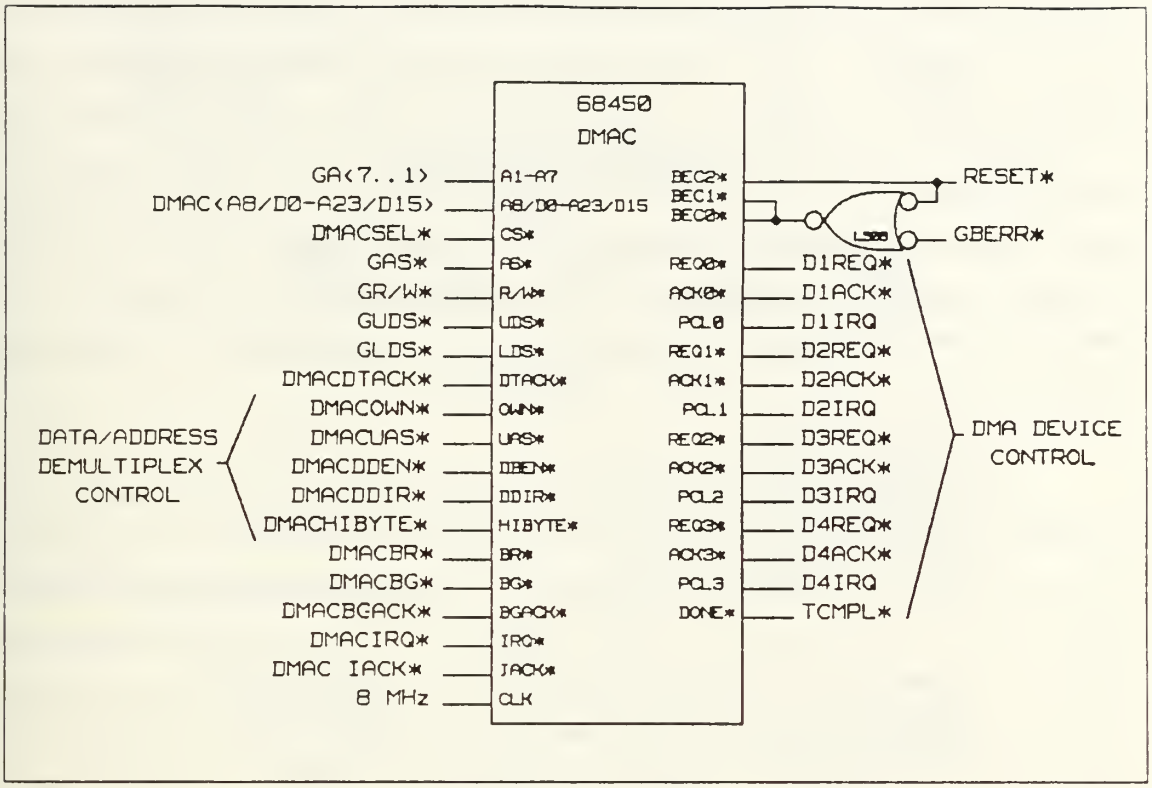


Figure 3.25 DMAC Signals.

upper 16 address lines are multiplexed with the data lines, DMAC<A8 D0-A23 D15>. The address data demultiplexing and global bus control line direction reversal when in the DMA mode is controlled by the the data/address demultiplex control signals in Figure 3.25. The demultiplex control signals are used by the DMAC bus control section to accomplish the demultiplexing and to control signal direction to correspond to the DMAC mode of operation.

The DMA devices (disk controllers) and DMAC are programmed by the CCU to move blocks of data between the disk drives and data buffer memory. Once the data transfer is initiated by the CCU, the disk controllers request service by asserting a request (D1REQ*-D4REQ*) to the DMAC. If the DMAC is not the current global bus MASTER, it will request control of the global bus. After becoming the global bus MASTER, the DMAC will assert an acknowledge (D1ACK*-D4ACK*) to the disk controller indicating that the requesting disk controller should put data on, or read data from, the global bus as determined by the direction of the GR/W* signal. The DMAC will continue this process until all active requests from the disk controllers are

satisfied and then release the global bus. During the last transfer in a block of data, the DMAC will assert TCMPL* with the acknowledge to indicate to the disk controller that the transfer is complete. After receiving the TCMPL* signal, the disk controller will generate an interrupt request (D1IREQ-D4IREQ) to the DMAC indicating that the status of the completed operation is ready to be read. The DMAC will pass the disk controllers interrupt to the CCU by asserting DMACIREQ* and provide a vector during the interrupt acknowledge cycle that identifies the disk controller that caused the interrupt. The CCU uses the returned vector to read the status of the interrupting disk controller to determine if the operation terminated correctly.

1. **DMAC Bus Control**

The DMAC bus control section is shown in Figure 3.26. This section is used to provide signal direction control and data/address demultiplexing corresponding to the active global bus MASTER. When the CCU is the active global bus MASTER, the signal flow is from the global bus to the DMAC. In this case the global address lines GA<7..1> are used for DMAC internal register selection and the multiplexed DMAC<A8/D0-A23/D15> lines are used as the data path. When the DMAC is the active global bus MASTER the signal flow is from the DMAC to the global bus. The DMAC control signals that direct the signal flow are:

- DMACOWN* - asserted when the DMAC is the active global bus MASTER and used to enable external address drivers and control signal buffers
- DMACUAS* - asserted when the DMAC is the active global bus MASTER and used to capture the value of the upper address lines on the multiplexed address data bus, DMAC<A8/D0-A23/D15>
- DMACDBEN* - used as the enable for external bidirectional buffers
- DMACDDIR* - used to control the direction of the external bidirectional data buffers, asserted if the transfer is from the global bus to the DMAC and negated if the direction is from the DMAC to the global bus
- DMACHIBYTE* - asserted when the DMAC is the active global bus MASTER and used to allow 8-bit devices to exchange data with memory on the upper byte (GD<15..8>) or lower byte (GD<7..0>) of the global bus in the implicit addressing mode.

The DMAC bus control circuit in Figure 3.26 consists of three functional sections:

- bidirectional control signal buffers
- address driver/latch
- bidirectional data buffers

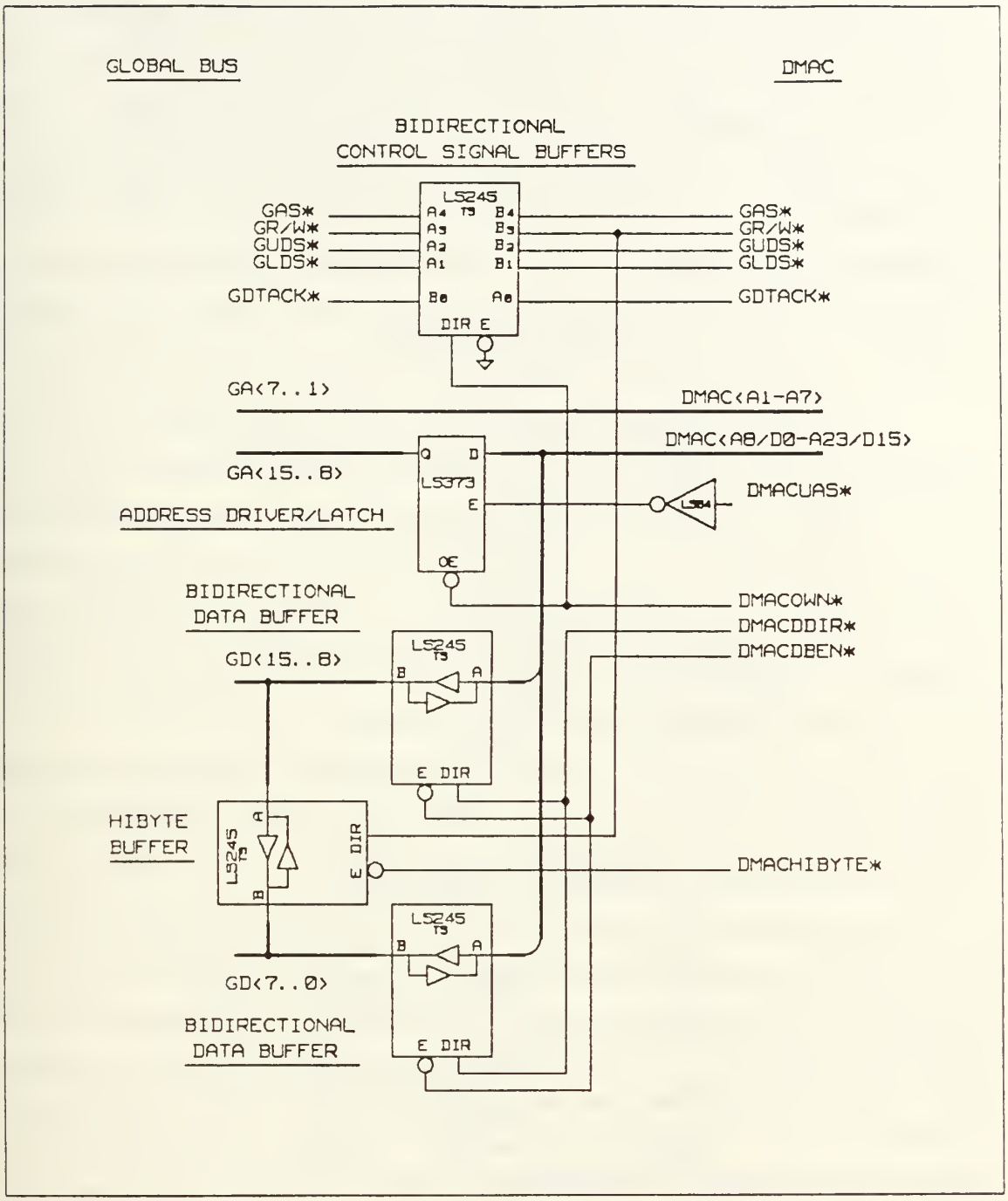


Figure 3.26 DMAC Bus Control.

The bidirectional control signal buffers are controlled by DMACOWN*. When the CCU is the active global bus MASTER, DMACOWN* is negated and the A inputs to the control signal buffers are passed to the B outputs, corresponding to the

CCU driving the controls and receiving the DMAC data transfer acknowledge. When the DMAC is the active global bus MASTER the situation is reversed. DMACOWN* is asserted and the B inputs to the control signal buffers are passed to the A outputs.

The address driver latch (LS373) demultiplexes the upper address lines from the data lines. As the active global bus MASTER, the DMAC will assert DMACOWN* which enables the outputs from the LS373. The DMAC then drives the multiplexed address data bus, DMAC<A8 D0-A23 D15>, with the upper address of the operand and asserts DMACUAS*. The upper address is latched on the rising edge of DMACUAS*. At this point the global address bus has a valid address, GA<7..1> driven by the DMAC directly and GA<15..8> driven by the address driver latch.

The bidirectional data buffers are enabled by DMACDBEN* with direction control from DMACDDIR*. The DMACDBEN* and DMACDDIR* signals are used for all transfers to or from the DMAC, regardless of who is controlling the bus. If the CCU is the active global bus MASTER, DMACDBEN* is asserted after GUDS* or GLDS* is asserted by the CCU with the directional control, DMACDDIR*, governed by the GR/W*. If the DMAC is the active global bus MASTER, DMACDBEN* is asserted after DMACUAS* is negated and before the DMAC asserts GUDS* or GLDS*, again DMACDDIR* is governed by GR/W*.

The DMACHIBYTE* signal is a special purpose signal used to adapt 8-bit devices to 16-bit word, byte addressed memories when implicit addressing is used in DMA operations. In implicit addressing, the DMAC provides the single byte address of the source or destination in memory and performs the transfer in a single bus cycle. To accomplish this with 8-bit devices such as disk controllers, requires an additional data path when even addresses are accessed. Even addresses will use the upper byte of the global data bus, GD<15..8>, but the disk controllers are on the lower byte of the global data bus, GD<7..0>. For implicit addressing, a path between the upper and lower bytes of the global data bus is provided by the HIBYTE buffer which is controlled by the DMACHIBYTE* and GR/W* signals. DMACHIBYTE* will be asserted by the DMAC when an 8-bit device is used in a transfer with an even address location. DMACHIBYTE* enables the HIBYTE buffer to gate the upper byte of the global data bus to the lower or vice versa depending on GR/W*. If GR/W* is high, for a memory read, the upper byte read from memory will be gated to the lower data bus byte and to the disk controller. If GR/W* is low, for a memory write, the lower byte read from the disk controller will be gated to the upper data bus byte and to memory. In explicit addressing the byte swapping is accomplished by the DMAC internally.

2. Global Bus Control

The global bus control section generates the signals required for the global bus MASTER to access all devices on the global bus. The control signals generated are standard 68000 type memory control signals. The global bus control section consists of a global memory control section and a global DTACK BERR generator section. The global memory control circuits are shown in Figure 3.27.

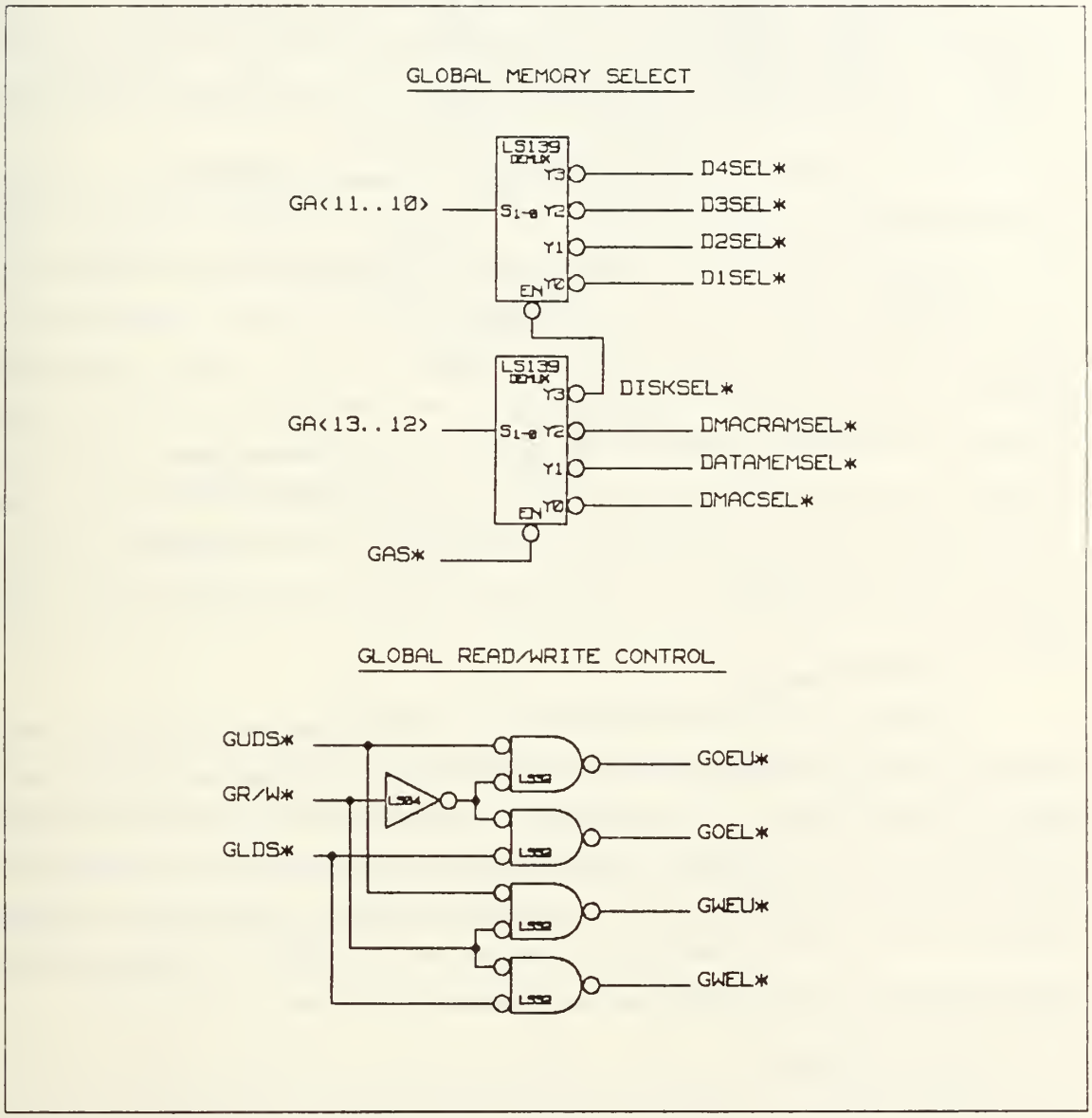


Figure 3.27 Global Memory Control.

The global memory select circuit decodes global address lines GA<13..12> to generate the chip selects for the DMAC (DMACSEL*), data buffer memory (DATAMEMSEL*), CCU DMAC control memory (DMACRAMSEL*), and disk controller select (DISKSEL*). The DISKSEL* is used as the enable for another level of global address decoding, GA<11..10>, which selects one of the four disk controllers (D1SEL*-D4SEL*).

The global read write control circuit generates the output enables (GOEU* and GOEL*) for read operations and write enables (GWEU* and GWEL*) for write operations. The output and write enables are asserted on a byte basis to allow byte and word operations with the selected device on the global bus.

The global DTACK BERR generator is shown in Figure 3.28. This circuit terminates global bus data transfers by generating the global bus data transfer acknowledge (GDTACK*) for normal termination or global bus error (GBERR*) when an unpopulated part of memory is referenced. This circuit uses a shift register as a timer to provide 250 nanosecond time intervals for data transfer acknowledge generation and a 2 microsecond time out for bus error generation. The dual ported memories, data buffer memory and CCU DMAC control memory, may be busy during a global bus access so the timer is inhibited by the wait signal (MEMWAIT*) asserted by the dual ported memories when they are busy. This allows the global bus access to resume without generating a bus error when the dual ported memories are no longer busy.

3. **Global Memory**

There are only two true memories in the global memory, data buffer memory and CCU DMAC control memory. The port A circuits of these dual ported memories were discussed in Figure 3.8 and Figure 3.17 respectively. The port B circuits for these memories are shown in Figure 3.29. The port B circuits are the same as the port A circuits except for the different signal names used to reflect the global bus as the signal source. The remaining global memory devices are the DMAC and disk controllers which are treated as memory but are not true memory devices.

F. DISK CONTROL

A block diagram of the major circuits in the disk control section is shown in Figure 3.30. The primary circuits are: DMA request delay, disk controllers, and disk interfaces. The disk controller and disk interface circuits will be presented only once because they are identical for each attached disk drive.

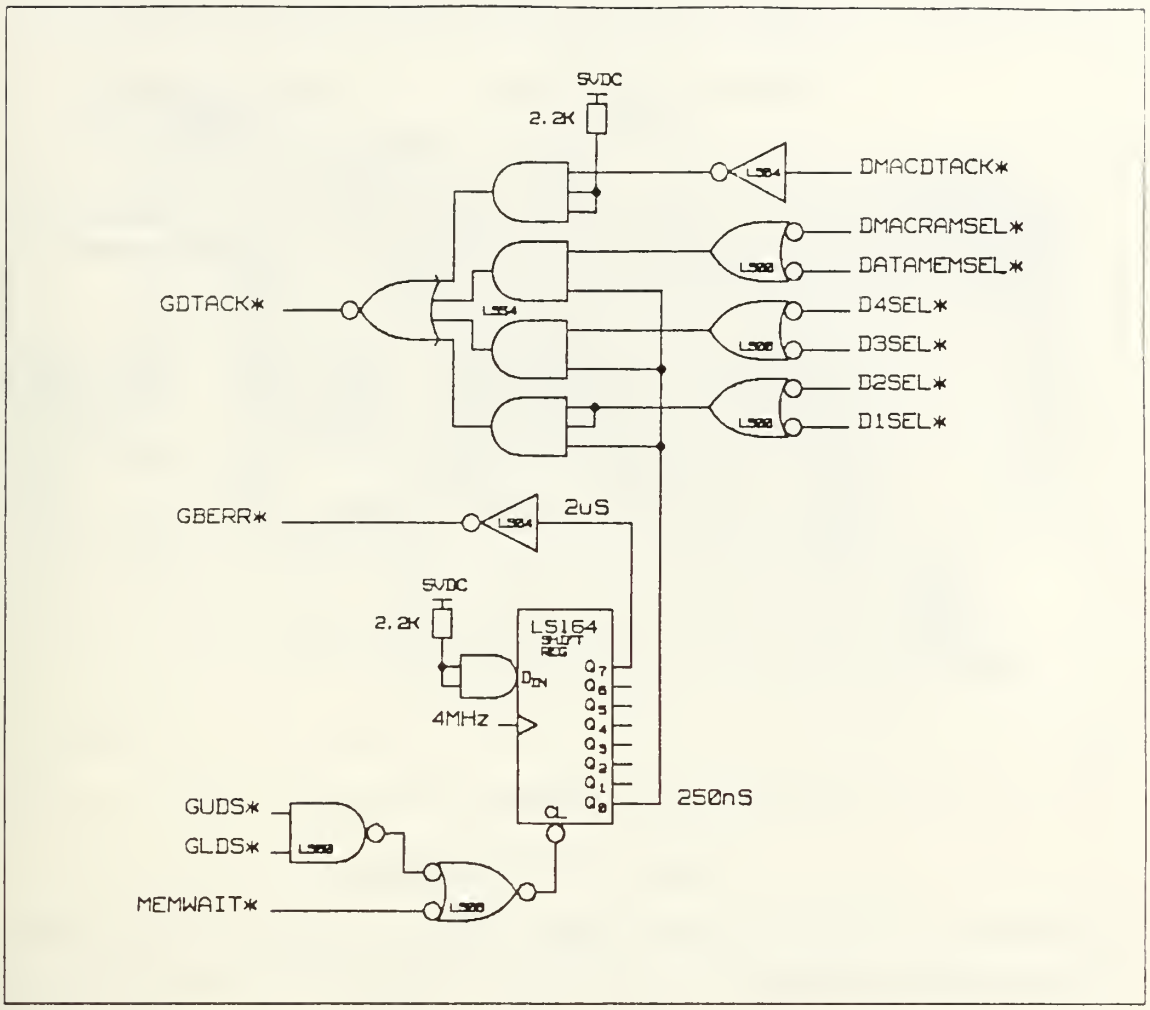


Figure 3.28 Global DTACK/BERR.

The CCU communicates with the disk controllers via the global bus. The disk controller's internal control and status registers appear to the CCU as byte memory locations in the global memory. The CCU initiates operations with, and checks status of, the disk controller by reading or writing to the disk controller's internal registers in a normal memory reference cycle.

The DMAC communicates with the disk controllers by using the global data bus, global read/write(GR/W*), and the DMAC DMA device control signals. This communication is not a standard 68000 memory reference as is the case with the CCU because the FDC uses an Intel type bus protocol with separate read and write lines. The DMAC will use implicit addressing for DMA transfers between memory and the disk controllers. This means that the global address bus and global memory control

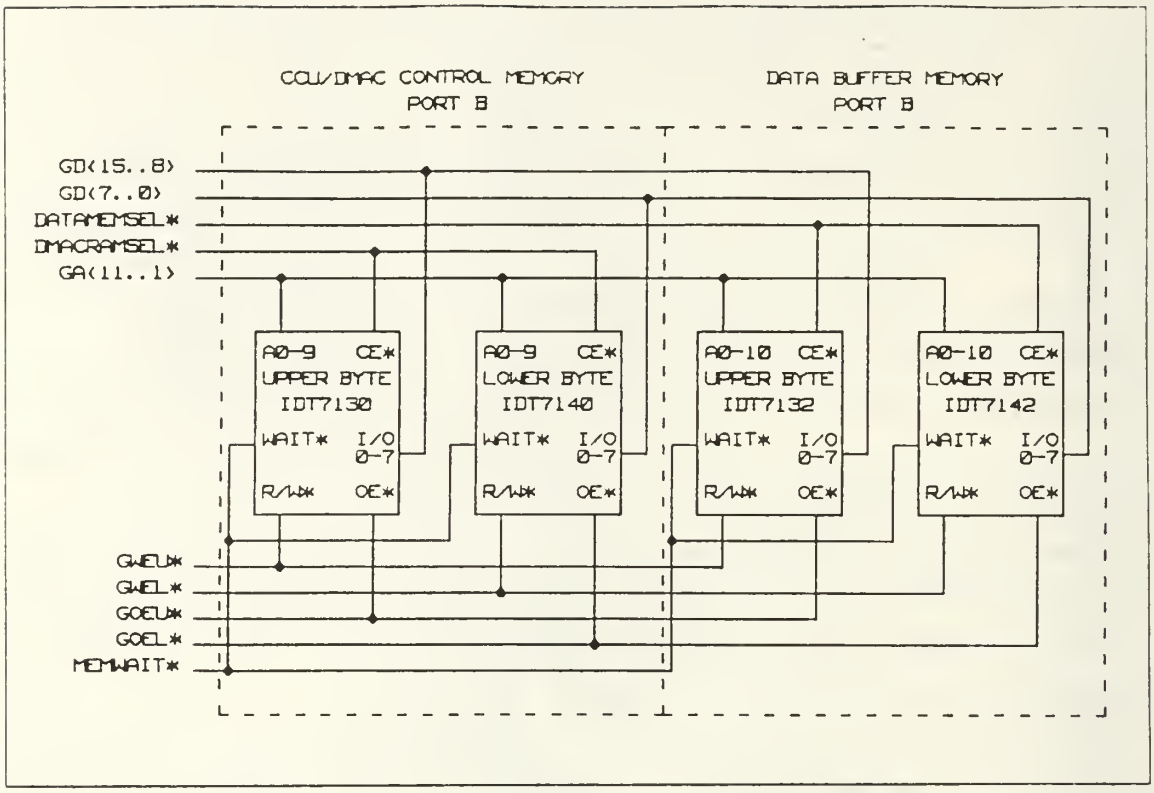


Figure 3.29 Global Memory.

signals will be used to control the memory operation while the DMAC DMA device control signals will be used to control the disk controllers. The DMAC DMA device control signals are sufficient for the control of the disk controllers during DMA operations; however, the disk controllers require additional circuits to generate the read and write signals needed for data direction control.

In CCU memory reference cycles to the disk controller, the GR/W* signal follows the standard 68000 memory reference protocol: GR/W* = low indicates that the CCU is writing to the disk controller, GR/W* = high indicates that the CCU is reading the disk controller. The disk controller reacts properly to the GR/W* signal by placing data on the global data bus during read cycles and taking data from the global data bus during write cycles.

In DMA operations with implicit addressing, the global address bus and memory control signals are referenced to the memory being accessed. In this case GR/W* = high indicates that the memory is being read by the DMAC and the data path is from memory to the disk controller. The disk controller will not interpret the GR/W* signal

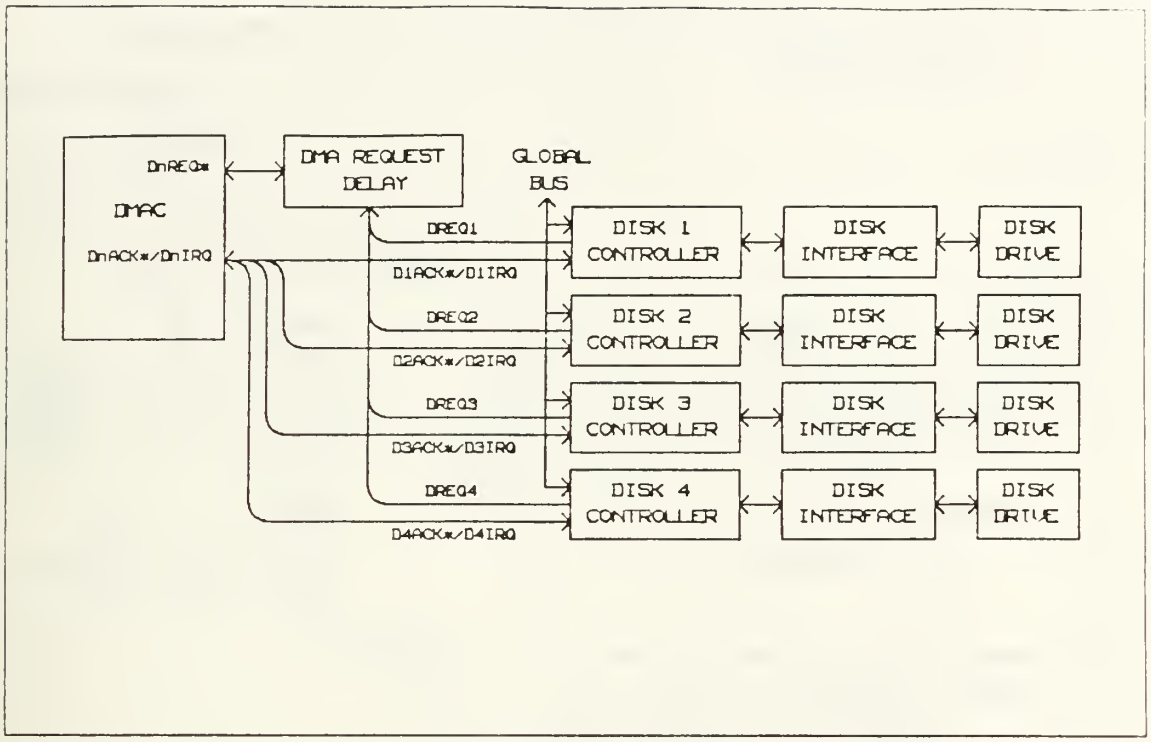


Figure 3.30 Disk Control Block Diagram.

correctly in this case unless a means is provided to reverse the sense of the GR W* signal in DMA operations using implicit addressing. The circuit that accomplishes the proper interpretation of the GR W* signal is part of the disk controller blocks in Figure 3.30.

1. Disk Controller

Each disk controller block in Figure 3.30 contains the circuit shown in Figure 3.31, where the n in signal names may be from 1 to 4 representing the possible disk controllers. This circuit is based on a Standard Microsystems Corporation 9268 floppy disk controller (FDC). The FDC can control up to four disk drives and can be configured to control 8 inch, 5 1/4 inch, or 3 1/2 inch disk drives with capacities to 720 kilobytes per disk. The configuration presented in Figure 3.31 is for standard 5 1/4 inch disk drives.

The signals on the right side of Figure 3.31 are the control signals exchanged with the disk interface circuit which controls the disk drive operation. The signals on the left side are the signals used by the CCU and DMAC in performing DMA operations.

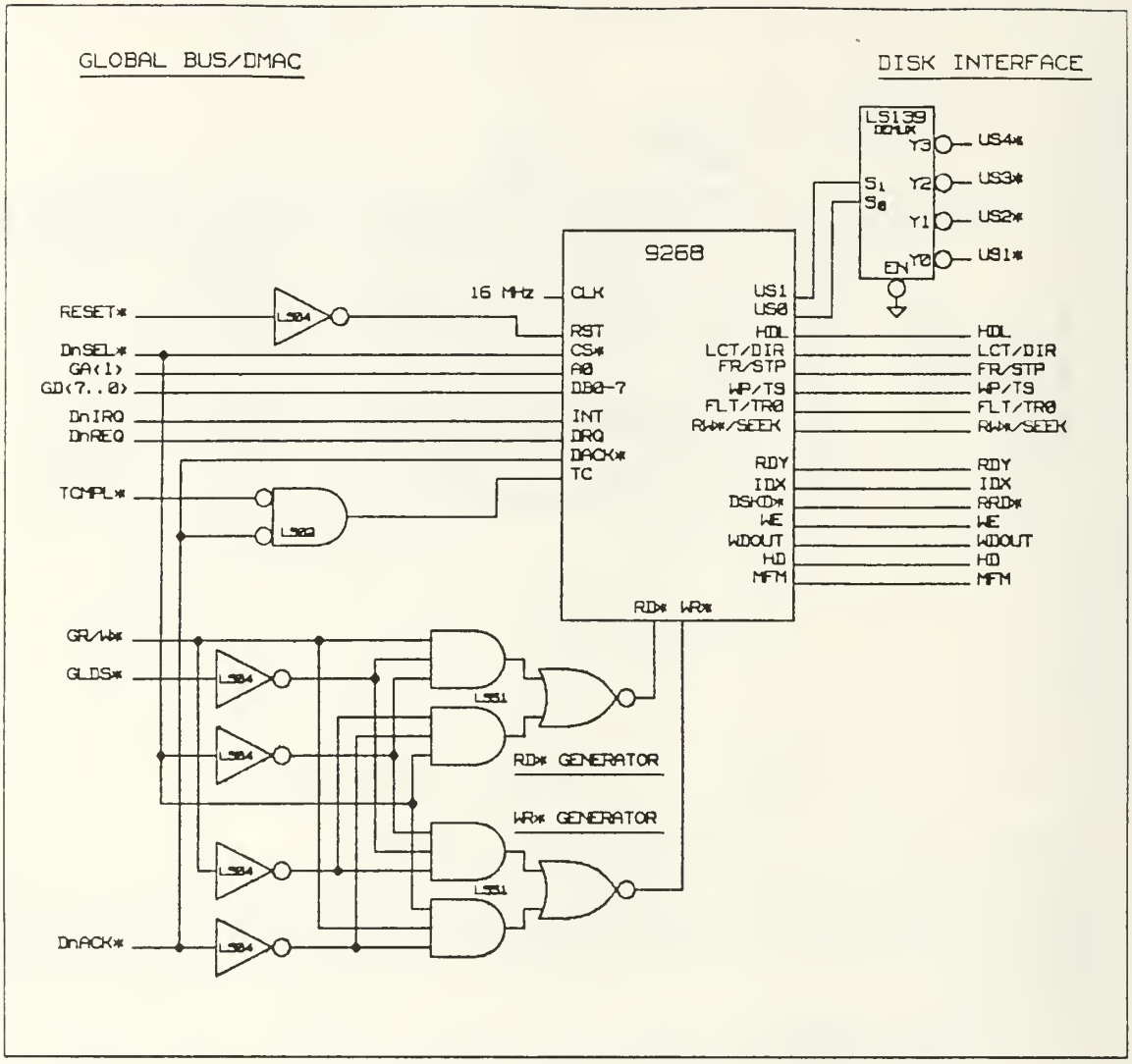


Figure 3.31 Disk Controller.

The FDC has two internal registers accessible by the global bus; the main status register and the data register. The CCU programs the FDC by writing command parameters in the FDC data register and checks status by reading the main status register. As previously discussed, the CCU operations on the FDC internal registers follow the standard 68000 memory reference protocol. The FDC register selection is controlled by global address bit 1 when the FDC chip select (DnSEL*) is asserted; GA < 1 > = low selects the status register, GA < 1 > = high selects the data register. In DMA operations when DnSEL* is not asserted, such as implicit addressed DMA, the data register will always be selected.

The RD* and WR* generators interpret the control signal inputs to determine the proper sense of the FDC read (RD*) and write (WR*) signals. The upper AND gate in each generator is used to generate the proper assertion level for a standard 68000 memory reference from the global bus. The lower AND gate in each generator is used for DMAC implicit addressed DMA operations.

Standard 68000 memory references from the global bus will have the FDC chip select (DnSEL*) and global lower data strobe (GLDS*) asserted. These signals enable the upper AND gates in the generators and allow the global read write (GR W*) signal to select the proper FDC read or write signal; GR W* = low asserts WR*, GR W* = high asserts RD*.

DMA operations with the FDC follow the three signal handshake protocol described for the DMAC. When the FDC is ready to read or write data it will assert a request (DnREQ) to the DMAC. The request does not go directly to the DMAC but must be delayed for a short period as will be described in the DMA request delay section. The DMAC will respond to the request, after becoming the global bus MASTER, with an acknowledge (DnACK*) indicating that the FDC should take data from, or put data on, the global bus. The request-acknowledge handshake takes place for each byte of data transferred. The final signal in the handshake is the transfer complete (TCMPL*) signal from the DMAC which is asserted with the acknowledge of the final byte to be transferred. The TCMPL* and DnACK* signals are combined to assert the terminal count signal (TC) which tells the FDC that the final byte transfer is in progress and will cause the transfer to terminate upon completion of the transfer. After the last byte is transferred, the FDC will generate an interrupt (DnIRQ) to the DMAC and follow the protocol described in the DMA control section. It is up to the CCU to read the status of the completed operation before initiating a new operation.

The above DMA protocol uses the GR W* signal to control data transfer direction. As previously discussed, the GR W* interpretation in implicitly addressed DMA operations is opposite to that of normal global bus memory references. For implicitly addressed DMA operations the GR W* sense is corrected by the lower AND gates in the RD* and WR* generators. In implicitly addressed DMA operations the FDC chip select (DnSEL*) will be negated, disabling the generator's upper AND gates, and the DMAC acknowledge (DnACK*) will be asserted. Asserting DnACK* enables the lower AND gates to correctly interpret GR W*; GR W* = low asserts RD*, GR W* = high asserts WR*, providing the opposite read write sense from normal global bus memory references.

The lower AND gates in the RD* and WR* generators are disabled by the FDC chip select (DnSEL*). This is not necessary if only implicitly addressed DMA operations are performed because DnSEL* and DnACK* will not be asserted together in implicit addressing. However, the DMAC can use explicitly addressed DMA operations which use the normal memory reference GR W* sense and asserts DnSEL* and DnACK*. Disabling the RD* and WR* generator's lower AND gates with DnSEL* will allow the proper RD* and WR* signals to be generated for explicitly addressed DMA operations. Explicitly addressed DMA operations are not the normal mode of DMA operations in the DCM because it takes twice as long to move a byte of data, two memory cycles, but explicitly addressed DMA operations are supported in case of future need.

The disk control signals shown in Figure 3.31 were previously defined in Chapter II as the standard control signals used by most disk drives. The disk control signals generated by the FDC are not all required by the 5 1/4 inch disk drives used in the DCM, but all of the disk control signals are sent to the disk interface circuit for buffering and demultiplexing to make them available in the event that different type disk drives are installed. This allows changing disk drive types by reconfiguring the disk drive connector to include the signals required by the particular disk drive installed.

All signals exchanged between the FDC and disk drive require buffering by receivers and transmitters because the FDC is not capable of directly receiving or driving these signals. Additionally, four of the disk control signals are multiplexed in accordance with the operation the disk drive is to perform. The multiplexed signals are logically grouped into read write operations (RW*) and seek operations (SEEK). The multiplexing is controlled by the RW* SEEK signal to produce the signals of Figure 3.32.

2. **Disk Interface**

The disk interface circuit provides disk drive control signal conditioning and demultiplexing between the FDC and disk drive. The disk drive control signal protocol calls for driving all signals with open collector drivers, with most signals asserted low. The disk interface circuit is shown in Figure 3.33.

Signals sent to the disk drive use the 7416 and 7417 open collector buffer drivers with terminating resistors supplied at the disk drive receivers. These drivers supply the required drive current and translate high asserted FDC signals to the low asserted level required by the disk drive.

FDC SIGNAL	SIGNAL GENERATED FOR DISK OPERATION READ/WRITE	SEEK	FDC SIGNAL DIRECTION
RW*/SEEK	RW*	SEEK	OUT
LCT/DIR	LCT	DIR	OUT
FR/STP	FR	STP	OUT
WP/TS	WP	TS	IN
FLT/TRØ	FLT	TRØ	IN

Figure 3.32 FDC Mulptiplexed Signals.

The disk drive uses open collector buffer drivers to send signals to the FDC. 74LS240 tristate receivers are used to receive the inputs from the disk drive. The incoming signals are terminated at the receiver inputs by 150 ohm resistors, the terminating resistors are not shown in Figure 3.33.

The 74LS240 tristate receivers also multiplex the incoming disk drive signals and demultiplex the FDC outgoing signals in Figure 3.32. The RW*/SEEK signal selects the proper transmitters and receivers for the chosen operation and disables (puts in a high impedance state) those transmitters and receivers not selected.

3. **DMA Request Delay**

In the prior discussion of the DMA protocol that is followed by the FDC, a need for delaying the FDC's DMA request (DnREQ) was mentioned. The FDC's DnREQ signal to the DMAC must be delayed in reaching the DMAC because the DMAC may respond too quickly for the FDC. This situation arises because the FDC asserts DnREQ 800 nanoseconds before data is ready for transfer, but the DMAC may start the transfer 375 nanoseconds after receiving DnREQ. An attempt to transfer data before the FDC is ready, less than 800 nanoseconds after DnREQ is asserted, will result in an error termination of the transfer.

The response time of the DMAC to DMA requests is determined by the state of the DMAC when the request is received. If the DMAC is not the global bus MASTER, it will take a minimum of twelve clock cycles, 1.5 microseconds at 8 MHz, to become the global bus MASTER and start the transfer. If the DMAC is already the

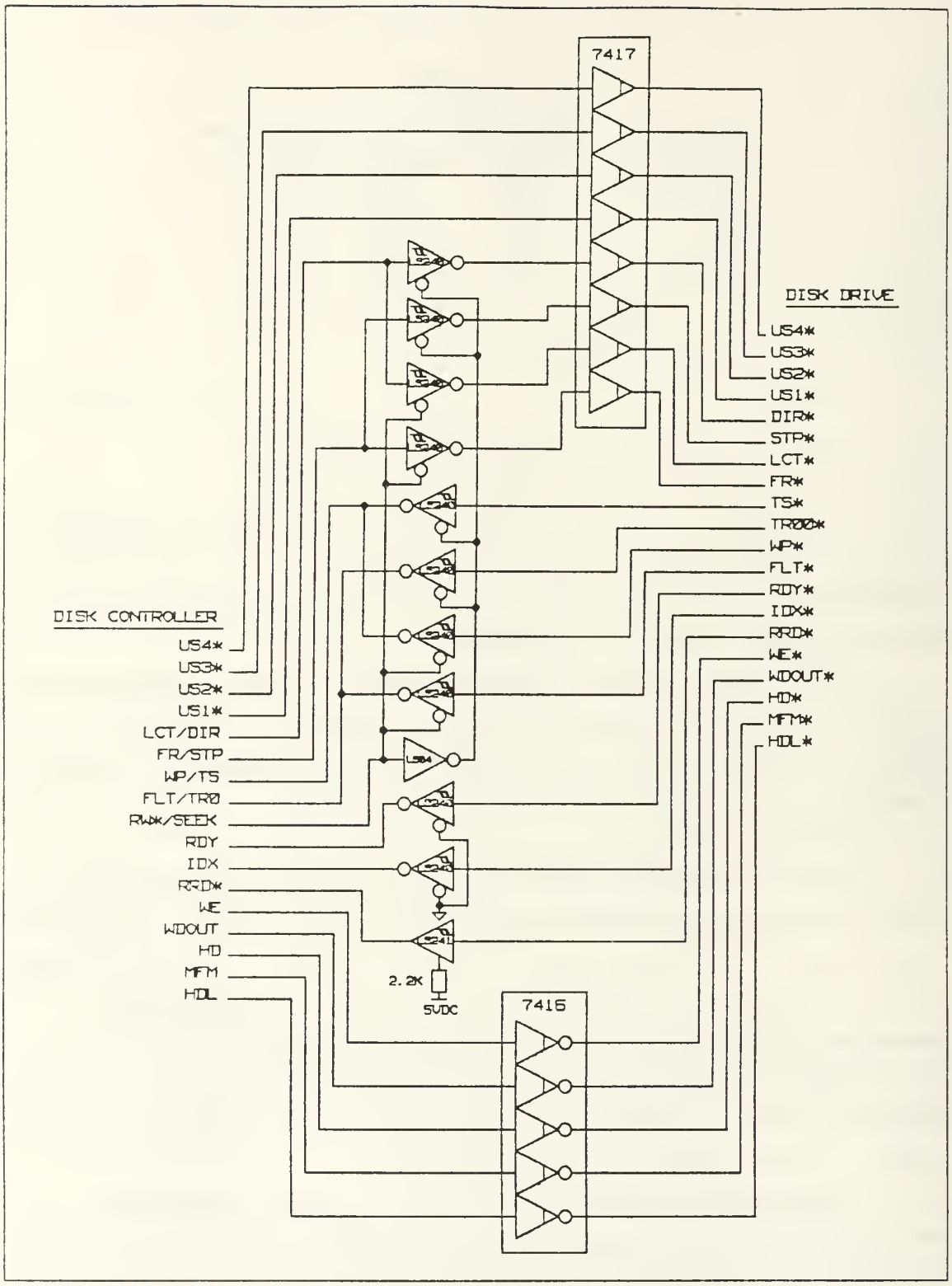


Figure 3.33 Disk Interface.

global bus MASTER, servicing another FDC when the request arrives, the DMAC may start the transfer in three clock cycles, 375 nanoseconds at 8MHz. This is possible because the DMAC will respond to a request received before S2 of the current DMA cycle immediately upon completing the current cycle. The time difference between S2 of the current cycle and S0 of the next cycle may be as short as three clock cycles for a DMA read operation.

From the above it is apparent that with more than one FDC attached to the DMAC there is a possibility of the DMAC responding to a DMA request before the FDC is ready. A 425 nanosecond delay of the DMA request from the FDC is required to ensure that the DMAC does not respond before the FDC is ready. The DMA request delay circuit that provides the required delay is shown in Figure 3.34.

The DMA request delay circuit generates the DMA request signal (DnREQ*) to the DMAC after delaying the DMA request (DnREQ) from the FDC. The delay for each DMA request is accomplished by two D flip-flops, with common clock inputs. The delay period is a minimum of one clock period and a maximum of two clock periods. The common clock input of 2 MHz provides a minimum delay of 500 nanoseconds and a maximum delay of 1 microsecond.

The first flip-flop will be set on the rising clock edge after DnREQ assertion. The first flip-flop's output is connected to the second flip-flop's input which will cause the second flip-flop to set on the next rising clock edge after setting the first flip-flop. Setting the second flip-flop asserts the DMA request (DnREQ*) to the DMAC. Minimum delay occurs when DnREQ is asserted one flip-flop set-up time prior to the rising clock edge that sets the first flip-flop. Maximum delay occurs when DnREQ assertion does not meet the flip-flop set-up time and must wait for the next rising clock edge to set the first flip-flop.

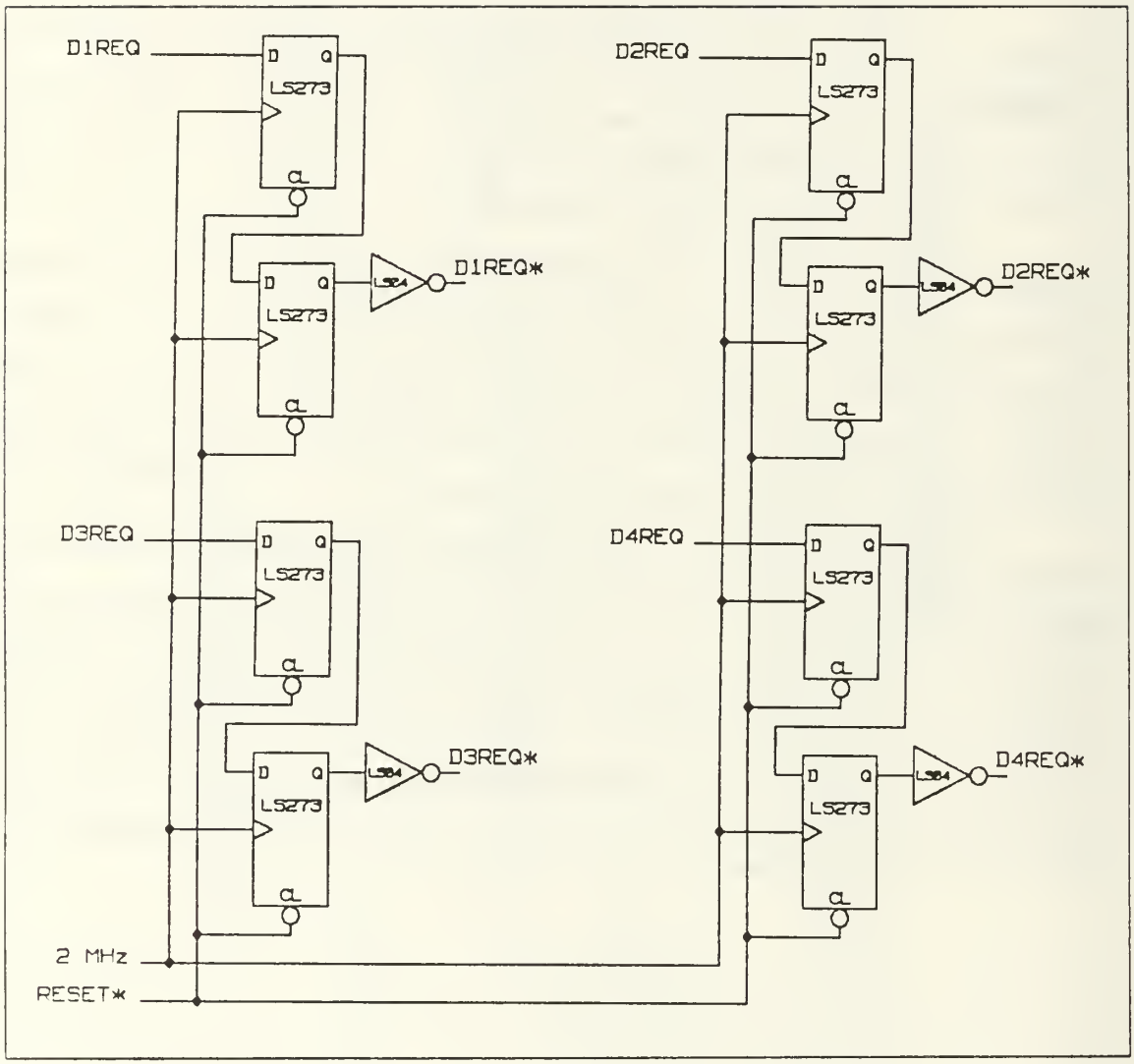


Figure 3.34 DMA Request Delay.

IV. SOFTWARE DEVELOPMENT

This chapter will present the interaction between the DCM's internal control software and the software of the user module (USER). The user module may be the host operating system (HOST) or any other module on the host bus. It is assumed that all USERs will get permission from the HOST before accessing the DCM to ensure data integrity and security. This allows multiple VME bus MASTERs to use the DCM provided the HOST coordinates the USERs activities.

The DCM's internal control software would consist of an onboard operating system (OBOS) that is capable of coordinating four external USERs and controlling the DMAC and four disk controllers. This is no small task and the development of the OBOS is beyond the scope of this paper; however, the basic interaction between the USER and OBOS will be described with flow charts.

The DCM appears to the USER as a segmented block of memory as described in Figure 3.5 where each disk has a dedicated host control buffer memory (CMD buffer) and data buffer memory (DATA buffer). Each disk also has a semaphore (BUSY flag) to indicate the disk's availability for use, a command present interrupt generator (CMD interrupt), and a status area (STATUS).

Access to the above memory locations for a disk is controlled by the BUSY flag associated with that disk. If a disk's BUSY flag is set, then that disk's CMD buffer, DATA buffer, STATUS, and CMD interrupt memory locations are not to be used by any USER other than the USER that set the BUSY flag. This is a software convention that must be followed by all USERs.

The format of a command in the CMD buffer is very flexible because there is only one hardware convention, the location of the CMD interrupt generators. The software designer is free to choose a command format based on the needs of the USER and the DCM's operating system. The following is a simple example of a command format to illustrate a possible format for a disk read or write command. The format is organized in bytes with byte 1 as the first byte in the CMD buffer:

- byte 1 - number of bytes in the command
- byte 2 - command completion interrupt vector number to be returned to the user upon command completion: the upper five bits are USER selectable and the lower three bits specify the the interrupt level
- byte 3 - operation code

- byte 4 - track number
- byte 5 - head number
- byte 6 - sector number
- byte 7 - data transfer length

The above format may repeat with a new operation code in byte 8 followed by more parameters, etc. The number and meaning of the parameters following the operation code may vary with the different operation codes because the OBOS would use a table look-up algorithm to decode the operation code and expected parameters.

The primary limitation on the number of operations that can be included in a single command to the DCM is the amount of DATA buffer memory per disk. Operations that use the DATA buffers, disk reads and writes, will be limited to the number of disk sectors that will fit into the DATA buffer; this means that a command would be limited to a single disk read or write if the DATA buffer is the same size as a disk sector.

A. USER COMMAND EXECUTION

A flow chart of the steps a USER follows in executing a command with the DCM is shown in Figure 4.1. The USER checks the availability of the selected disk (Dn, n from 1 to 4) with an indivisible read-modify-write cycle to the Dn BUSY flag, setting the Dn BUSY flag, if it was not set. If the BUSY flag was previously set, the new USER must wait for the disk to be released. This ensures only one USER at a time gains access to the disk. If a write operation is to be performed, then the data is written into the Dn DATA buffer. The command is then written into the Dn CMD buffer, and the final step is writing to the Dn command interrupt generator, generating the Dn CMD interrupt to notify the OBOS that a command is present. After the command is issued, the USER continues processing until the DCM responds with a command completion interrupt signaling the USER to terminate the command.

The DCM will notify the USER of the command completion by generating the interrupt and returning the vector number specified in the command. The USER reads the Dn STATUS to determine if the command was successful. For a read operation the data is read from the Dn DATA buffer. The USER indicates that the results, status or data, have been read and releases the CMD buffer and DATA buffer by writing to the command interrupt generator which generates the CMD interrupt to the OBOS. At this point the USER's command is complete, but the Dn BUSY flag is still set.

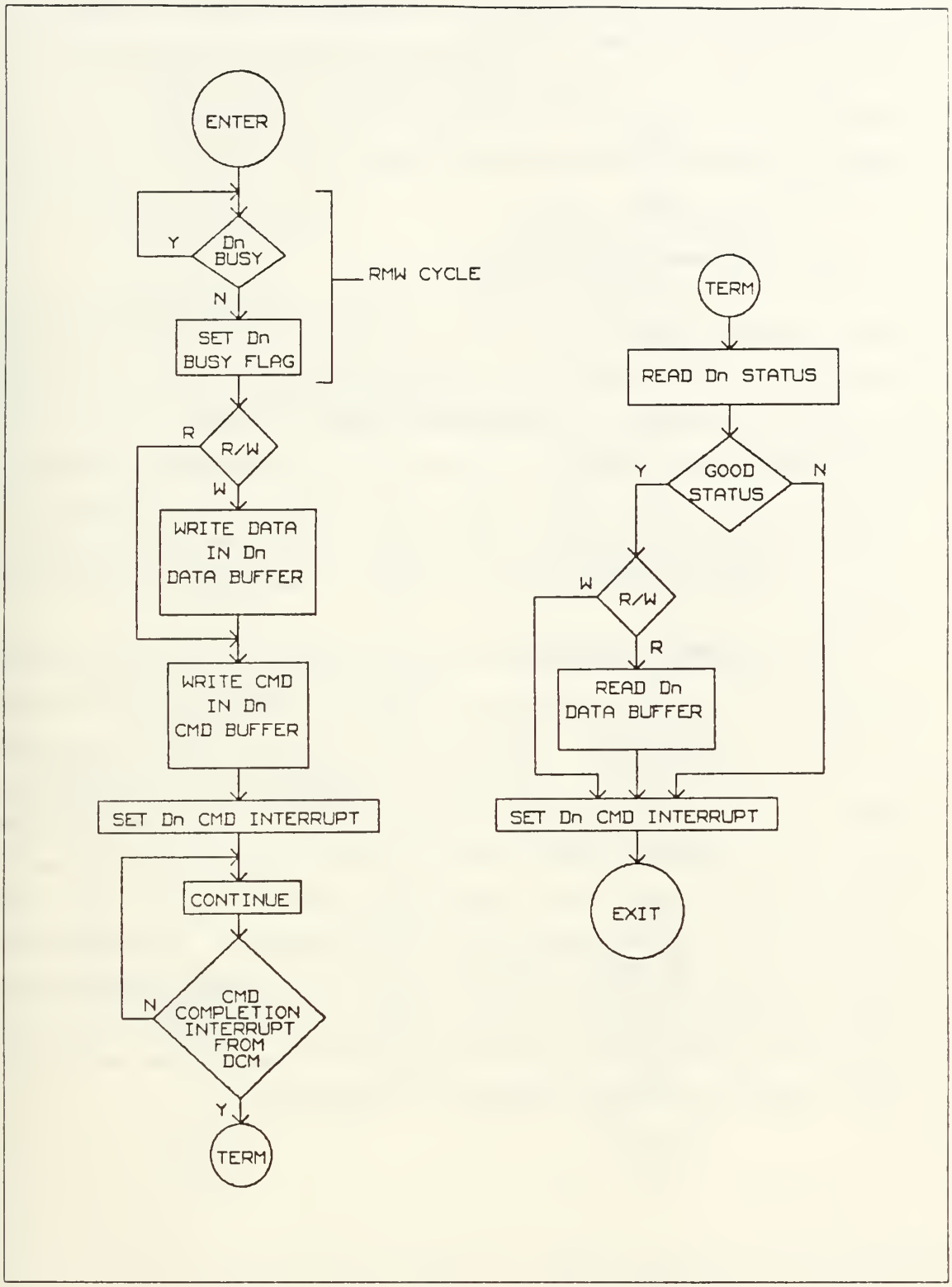


Figure 4.1 USER Command Execution Flow Chart.

The Dn BUSY flags are cleared by the OBOS. This allows the OBOS to take a disk off-line due to a failure or for multidisk operations such as copying from one disk to another disk. The disk copy command can be issued to either the source or destination disk and the DCM can capture the remaining disk when it becomes available by either not clearing the BUSY flag if it was in use or by immediately setting the BUSY flag if it is idle.

B. DCM COMMAND EXECUTION

The steps followed by the OBOS in initiating a command received from the USER are shown in Figure 4.2. When the Dn CMD interrupt is recognized, the OBOS locks out further CMD interrupts until the current command is started. The command is decoded and checked for validity with invalid commands causing an error termination indicated by an error status code. The decoded command parameters are passed to software modules that program the DMAC and disk controller for the requested operation. The OBOS enables CMD interrupts and continues processing until the command is complete which is indicated to the OBOS by an interrupt from the DMAC.

After the OBOS recognizes the command completion interrupt from the DMAC, the OBOS locks out further interrupts and checks the command completion status of the DMAC and disk controller. The results of the command completion status check will cause the OBOS to enter a normal or error status code in the Dn STATUS for the USER to check. The OBOS then notifies the USER that the command has completed by generating an interrupt on the host bus and returning the vector number specified in the command. The OBOS enables interrupts and continues servicing other USER's commands until the USER indicates with a CMD interrupt that the command results have been read. At this point the OBOS clears the command status and determines if the disk can be released for use by another USER. If it can, the OBOS releases the disk by clearing the Dn BUSY flag, otherwise the Dn BUSY flag is left set and the disk is used by the OBOS for another operation, such as a disk copy.

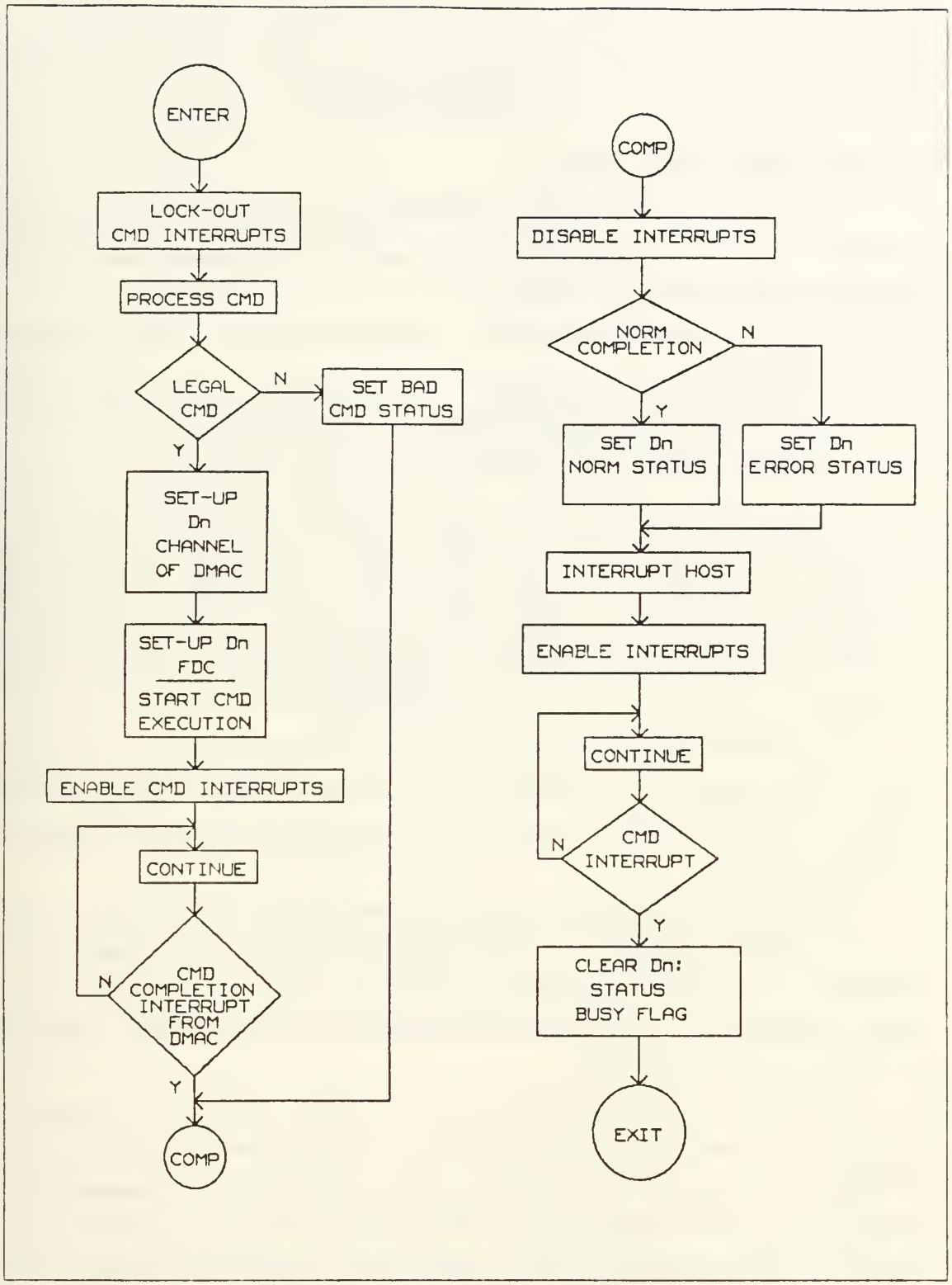


Figure 4.2 DCM Command Execution Flow Chart.

V. CONCLUSION

A. SUMMARY OF RESULTS

The principal goal of this thesis was the design of a flexible hardware kernal for a multidisk control module (DCM) which supports concurrent disk operations. Flexibility in three areas was desired:

- host bus interface - the ability to easily adapt to a variety of host bus architectures
- host operating system interface - compatible with most modern operating systems and easily integrated into an existing operating system
- disk drive interface - easily adaptable to a variety of disk drives.

The flexibility objectives of this thesis were met by using a modular design with the interface dependent hardware contained in separate modules and isolating these modules from the internal control modules. The host bus interface is isolated by dual ported buffer memories with the bus dependent hardware contained in the host bus control module. The disk drive interface is isolated by the disk controllers with the disk drive dependent hardware in the disk interface modules.

An added benefit of the modular design is device manufacturer independence. Each hardware module has a well-defined and relatively simple interface to adjoining modules. This should make it a simple matter to use different devices in implementing a module's function.

Concurrent disk operations are accomplished by using a separate disk controller for each installed disk drive. Each disk controller is capable of controlling up to four disk drives, but concurrent operation of all four disk drives is not possible. The DCM design will allow up to four disk drives per disk controller which means a maximum of 16 disk drives may be installed.

The DCM architecture separates the data transfer path from the control path. This allows the control functions to operate at a different speed than the data transfer functions. The data transfer functions can be optimized to accommodate the data transfer rate of the installed disk drives without affecting the control function operation. The net result is that a relatively slow microprocessor can be used for overall control and a fast direct memory access controller can be used to increase data transfer rates.

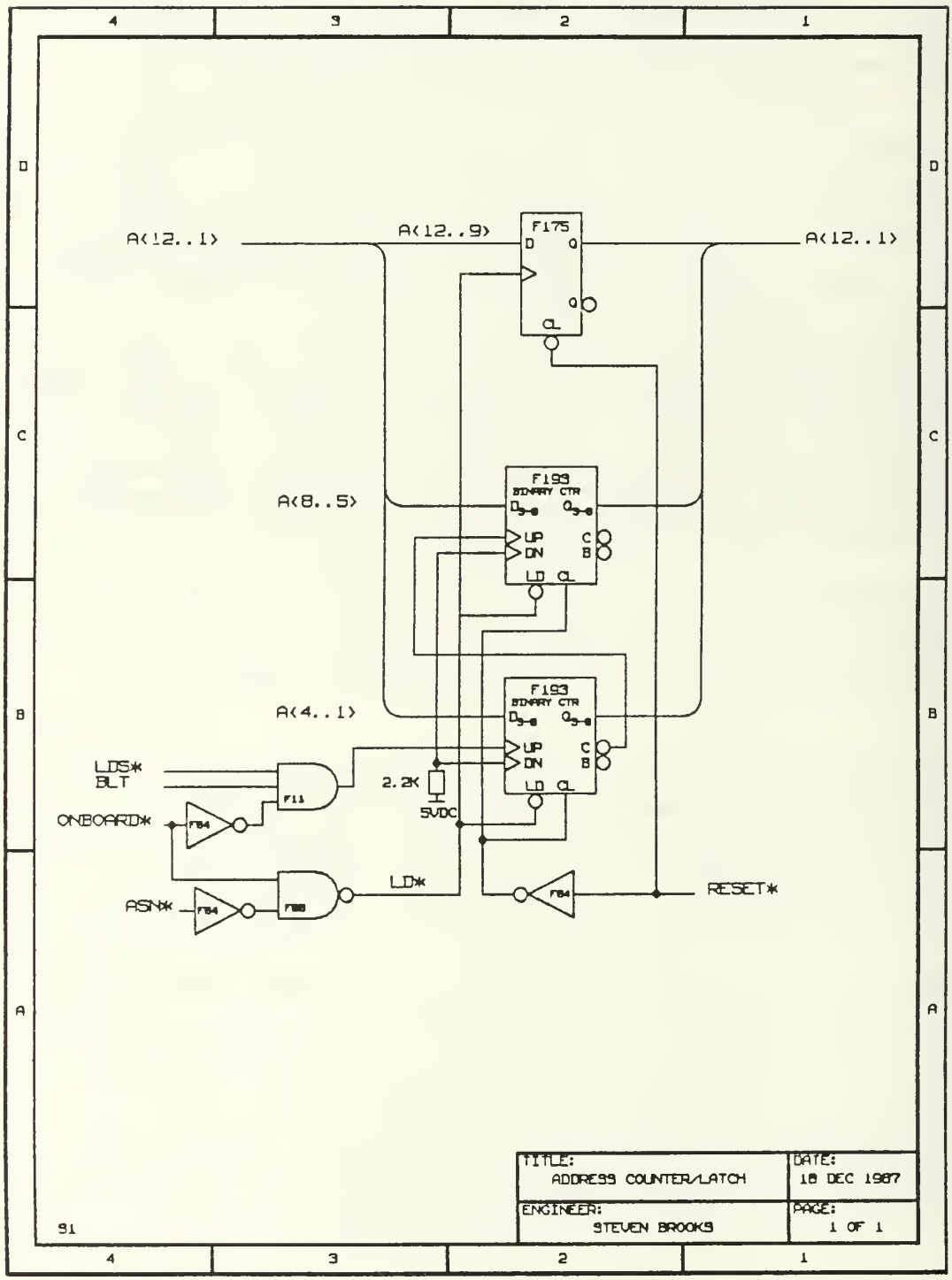
The DCM could not be exercised as part of a host system because a VME bus based host was not available. Such a host is being developed in another thesis and will provide a test vehicle for the DCM. The DCM data transfer circuits were tested with Tandon Corp. model TM100-2 and TEAC model FD55BV disk drives. The circuits operated satisfactorily in all modes with only minor disk interface adjustments necessary when switching between the two disk drives.

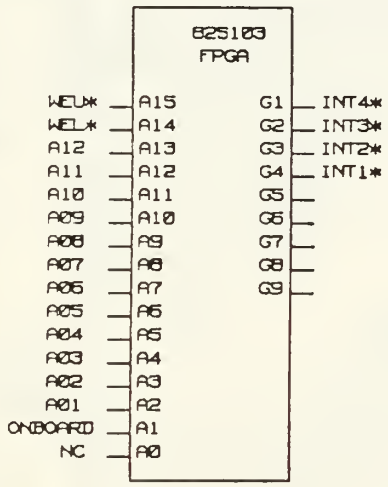
B. RECOMMENDATIONS FOR FUTURE RESEARCH

Two areas require further work: development of efficient software to optimize operation and investigation of methods to use non-DIP devices in prototyping. Developing the internal control software required to fully exploit the hardware capabilities will be a major task. The software required to interact with multiple users and efficiently control concurrent data transfers is a relatively complex onboard multiuser multitasking operating system.

Physical device size and limited prototype board space was a major problem during hardware development. The prototype board was limited to accepting only standard DIP devices which do not use area efficiently. For a large system, such as the DCM, a means of prototyping with the more area efficient shrink-DIP, PLCC, and PGA device packages should be developed.

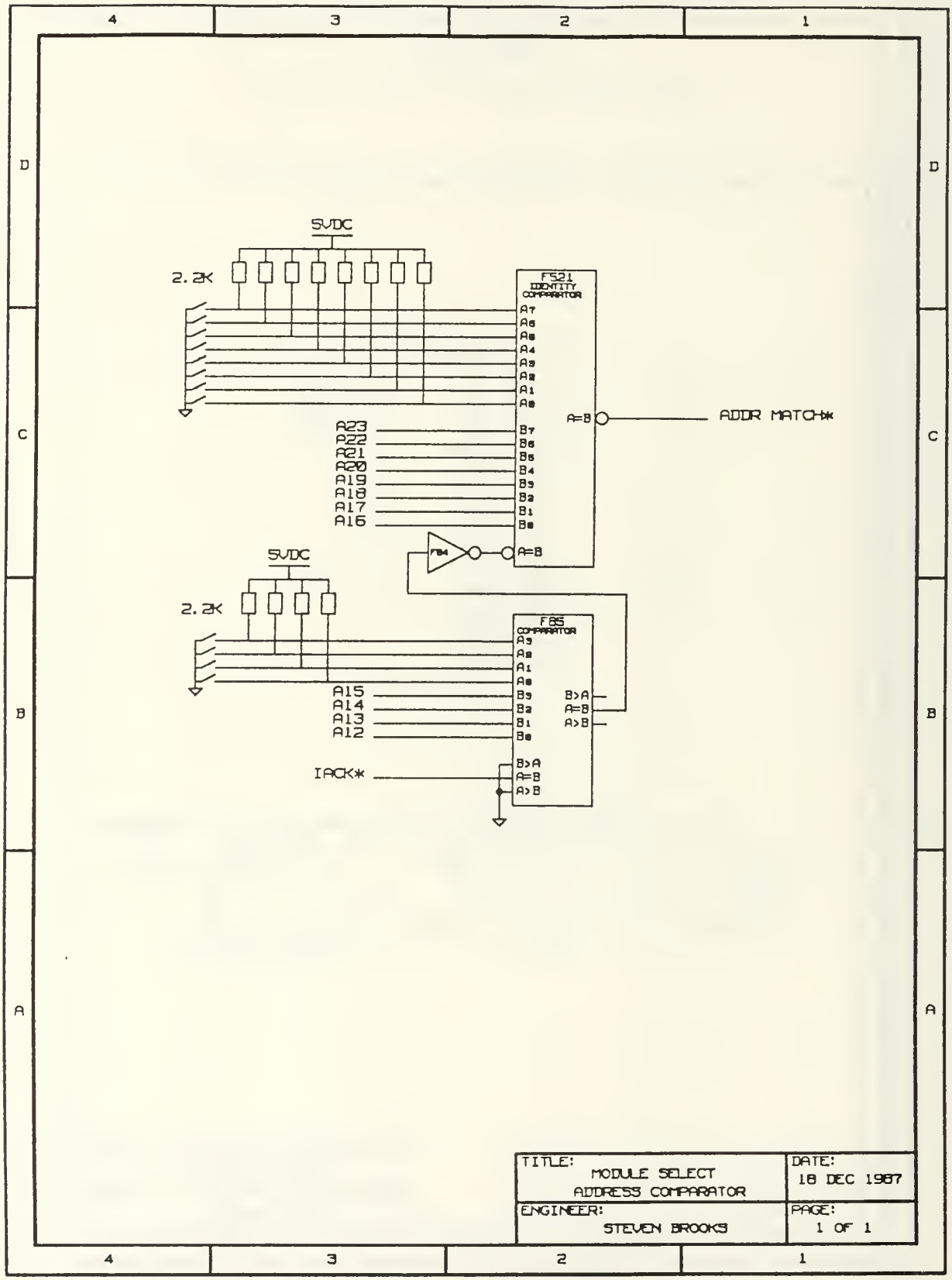
APPENDIX
FUNCTIONAL BLOCK SCHEMATICS





	WEU*	WEL*	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	ONBOARD
INT4*	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
INT3*	1	0	0	0	1	1	0	0	0	0	0	0	1	0	1
INT2*	0	1	0	0	1	0	0	0	0	0	0	1	0	1	1
INT1*	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1

TITLE: COMMAND INTERRUPT ADDRESS DECODER	DATE: 18 DEC 1987
ENGINEER: STEVEN BROOKS	PAGE: 1 OF 1


4
3
2
1
D
C
B
A
5VDC
2.2K
F521
IDENTITY
COMPARATOR
A7
A6
A5
A4
A3
A2
A1
A0
A=B
ADDR MATCH*
A23
A22
A21
A20
A19
A18
A17
A16
B7
B6
B5
B4
B3
B2
B1
B0
A=B
5VDC
2.2K
F85
COMPARATOR
A3
A2
A1
A0
B3
B2
B1
B0
B>A
A=B
A>B
A15
A14
A13
A12
IACK*
B>A
A=B
A>B
TITLE:
MODULE SELECT
ADDRESS COMPARATOR
DATE:
18 DEC 1987
ENGINEER:
STEVEN BROOKS
PAGE:
1 OF 1

LIST OF REFERENCES

1. Fischer, W., "IEEE P1014-A Standard for the High-Performance VME Bus," *IEEE Micro*, v. 4, pp. 31-40, February 1985.

2. Dawson, W.K. and Dobinson, R.W., "A Framework for Computer Design," *IEEE Spectrum*, v. 23, pp. 49-54, October 1986.

3. Borrill, P.L., "Microstandards Special Feature: A comparison of 32-Bit Buses," *IEEE Micro*, v. 5, pp. 71-79, December 1985.

4. Bach, M.J., *The Design of the UNIX Operating System*, Prentice-Hall, Inc., 1986.

5. Tanenbaum, A.S., *Operating Systems - Design and Implementation*, Prentice-Hall, Inc., 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	NASA - Johnson Space Center Attn: Dr. Larry W. Abbott Mail Stop EH431 Houston, TX 77058	1
5.	Professor Frederick W. Terman, Code 62 TZ Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6.	COMSPAWARSYSCOM PMW PMA-159 ATTN: LCDR Steven L. Brooks Washington, D.C. 20363-5100	1
7.	LT David M. Sendek SMC 2783 Naval Postgraduate School Monterey, CA 93943-5000	1

Thesis
B809442 Brooks
c.1 The design of an intelligent multidisk control model for VME bus based systems.